Specific Substring Problem: an application in bioinformatics

Lucas B. Rocha[1*], **Said Sadique Adi**[1], **Eloi Araujo**[1]

[1]FACOM – Universidade Federal do Mato Grosso do Sul (UFMS)
Campo Grande – MS – Brazil

lucas.lb.rocha@gmail.com, {said,feloi}@facom.ufms.br

Abstract. *Given two sets of sequences A and B, the Substring Specific problem is to find all minimum substrings in A having distance at least k for each subsequence in B. This work addresses three new implementations for the Maaß algorithm when the Hamming distance is considered: a naive cubic-time algorithm and two quadratic-time algorithms. We run tests to compare the running time of these implementations and another recently described algorithm implementation that uses the edit distance. In addition, we conducted preliminary testing on a large Tara Ocean database, looking for efficient and effective strategies for finding unique sequences in a set of sequences comparing with the other.*

1. Introduction

As a result of the advancement of DNA sequencing technologies, there are currently a large number of sequenced genomes. Tara Expedition were journeys where an international group of researchers sequenced the metagenome of over 35,000 different samples from 210 different regions in the world, resulting in a total of 7.2 terabytes of genomic data [Pesant et al. 2015, Bork et al. 2015]. This large amount of data needs now to be processed in searching for meaningful biological information. In this context, an important computational problem is determining markers, that are substrings from a set of sequences that do not occur on sequences of other sets, that is, small substrings found only in the former set. This is the Specific Segment Problem, proposed in [Gusfield 1997].

Other applications for determining markers include finding specific regions for primer design in PCR technology [Montera and Nicoletti 2008], and specific organisms in metagenomes that live in a given diseased tissue for the early diagnosis of cancer [Zitvogel et al. 2018, Shigefuku et al. 2017].

We formulate this problem in a simpler way, considering only two sets of sequences, that is, the problem of finding all the minimum substrings in a set of sequences A that do not appear in another set B. The time required and the quality of the markers found depend on how to define the distance function. Furthermore, since two equal sequences may have few differences due to some mutations or read errors, we consider only a pair of sequences having a distance greater than or equal to a given integer threshold k.

Dobre [Dobre 2017] implemented two versions of an algorithm described by Gusfield [Gusfield 1997] that uses edit distance, and spends time $O(n^3)$ in the worst case. Maaß [Maaß 2003] also described an algorithm that uses Hamming distance and spending time $O(n^2)$. The Hamming distance seems to be, in addition to being easier to calculate, a natural measure of similarity in many biological applications [Lanctot et al. 2003].

*This work is partially sponsored by UFMS and CAPES (scholarship 51001012028D6).

In this work, we describe three implementations of two different algorithms to solve the considered problem using Hamming distance: first, one that implements a naive strategy and, then, two versions of the Maaß's algorithm that use two different data structures; we compare the time spent by each of the three implementations and by that from Dobre's work [Dobre 2017]. In addition, we perform tests with two different samples obtained from the Tara project database, showing the time spent processing them.

This work is organized as follows: in Section 2, we describe the algorithms for the problem considering edit and Hamming distance, and we present the time complexity of each one. In Section 3, we show the practical results comparing our results to those from Dobre [Dobre 2017]; moreover, for two samples of Tara project database, we show the results regarding the time spent in simplified experiments using small subsets of two different samples. This was done to estimate the expected total spent time when comparing whole samples. Finally, in Section 4, we discuss the results and a strategy that we intend to adapt and follow in order to make this work feasible in practice.

2. Preliminaries

An *alphabet* Σ is a finite set of symbols. We denote a sequence s over Σ by $s_1 s_2 \dots s_\ell$ where each symbol $s_i \in \Sigma$. We say the *length* of s, denoted by $|s|$, is ℓ. We say that the sequence $s_i s_{i+1} \dots s_j$, with $i \geq 1$ and $j \leq |s|$ is a *substring* of s and we denote it by $s[i, j]$. A substring $s[1, j]$ is called a *prefix* of s.

Given a distance function, we say that a string s is a *minimal* string that have distance at least k to any string in a set of sequences A if the distance between s and any substring of A is greater than or equal k and, for any prefix s' of s with $|s'| < |s|$, there is a substring t in A such that the distance between s' and t is smaller than k.

Problem 1 (Specific Substring Problem — SSP-$\mathcal{D}$) *Given two sets A and B of sequences such that $|A| = M$, $|B| = N$, find all the minimal substrings in A that have distance at least k to any substring in B for some chosen distance function $\mathcal{D}$.*

The *edit distance (ED)* between two sequences is the minimum number of edit operations *insertions*, *deletions* and *substitutions* required to transform one sequence into another. In [Dobre 2017], Dobre implements an algorithm, called here ***KED,*** described by Gusfield [Gusfield 1997] that solves the SSP-ED in $O(MN \cdot n^3)$ time where n is the length of the longest sequence in $A \cup B$.

Given two n-length sequences s and t, the *Hamming distance (HD)* $d_H(s, t)$ between s and t is defined as

$$d_H(s, t) = \sum_{i=1}^{n} |s_i \neq t_i|,$$

where $|s_i \neq t_i|$ is equals to 1 if $s_i \neq t_i$, and is equal to 0 otherwise.

The *length of the smallest prefix* of s and t with k differences under the Hamming distance is denoted by $p(s, t, k)$ and it can be found by traversing s and t at the same time from left to right until we found k differences; if this number doesn't exist, we define $p(s, t, k) = \min\{|s|, |t|\} + 1$. This process can be done in $O(|s| + |t|)$ time.

Gusfield [Gusfield 1997] describes a naive algorithm for a restricted version of Problem SSP-HD. We implement and call it ***KHD1*** algorithm. This restricted version of

Problem 1 considers $|A|$ and $|B|$ having just one sequence each, saying s and t. Then, this naive algorithm has as input two sequences s and t, where $|s| = m$, $|t| = n$, and an integer k, and computes $r[i] = \max_j\{p(s[i, m], t[j, n], k)\}$ for each i. Each $r[i]$ can be computed in $O(mn)$ time. Thus, this algorithm is computed in $O(m^2n)$ time, i. e, $O(n^3)$ if we consider $n = m$.

For the same restricted version, we also implemented a faster algorithm described by Maaß [Maaß 2003]. This algorithm is summarized in Algorithm 1 and the corresponding implementation is called *KHD2*.

Algorithm 1 [Maaß 2003]

1: $r[i] \leftarrow 0$ for $i = 1, \ldots, m$
2: **for** all $(i, j) \in \{1, \ldots, m\} \times \{1, \ldots, n\}$, such that $i = 1$ or $j = 1$ **do**
3: $\quad max \leftarrow \min\{m, i + n - j\}$
4: $\quad d \leftarrow p(s[i, m], t[j, n], k - 1)$
5: $\quad L \leftarrow 0$
6: $\quad$ **while** $i + L \leq max$ **do**
7: $\quad\quad$ **while** $d + i + L < max \wedge\ s_{d+i+L} = t_{d+j+L}$ **do** $d \leftarrow d + 1$
8: $\quad\quad$ **if** $max \geq d + i + L$ **then** $d \leftarrow d + 1$
9: $\quad\quad$ **while** $TRUE$ **do**
10: $\quad\quad\quad$ **if** $r[i + L] < d$ **then** $r[i + L] \leftarrow d$
11: $\quad\quad\quad L \leftarrow L + 1$
12: $\quad\quad\quad d \leftarrow d - 1$
13: $\quad\quad\quad$ **if** $i + L > max\ \vee\ s_{i-1+L} \neq t_{j-1+L}$ **then break**
14: **return** r

The number of pairs (i, j) where $i = 1$ or $j = 1$ is $n+m-1 = O(n+m)$. The algorithm spends time $O(\min\{n, m\})$ to compute k differences between the $O(\min\{n, m\})$ pairs of sequences in each iteration. Thus, Maaß algorithm spends $O((n + m)^2)$ time, i. e, $O(n^2)$ if we consider $n = m$.

The Longest Common Extension (LCE) of two strings s and t is defined as the length of the longest s and t common prefix. Maaß [Maaß 2003] claims that, since LCE can be found in constant time by using a suffix tree, this structure can be used for executing Line 7 in constant time speeding up the algorithm in practice, although the theoretical complexity is still $O(n^2)$. We implemented the algorithm considering this improvement and we call it *KHD3*.

3. Experiments and results

The experiments reported in Section 3.1 were performed in a server Intel Xeon(R) E5-4650 2.7 GHZ, with 20 MB of cache memory, 95 GB of RAM memory and 8 Processing cores and those reported in Section 3.2 were performed on a cluster with 40 nodes with processors Xeon(R) CPU X3440@2.53GHz of 4 cores, 4 GB of RAM per node. In Section 3.1, we compare the running time of KHD1, KHD2, KHD3 and KED that are implemented in C++. The goal of the test is to find fast algorithms and good parameters for using in the fastest strategies for solving SSP. In Section 3.2, we use the best strategies to estimate the required time to compare two samples from Tara project sequences.

3.1. Tests with Homo sapiens sequences

In order to run the experiments with real data, two sets A and B including sequences obtained from the *Homo sapiens* database were used [Dobre 2017]. Set A includes three sequences, namely $hsarhgdig$ (4398 characters), $hsankr10$ (38530 characters) and $hsaff4$ (90284 characters), and the set B includes 7 sequences, namely $hsascl2$ (4455 characters), $hsankrd43$ (5457 characters), $hsa1bg$ (8694 characters), $hsalg10b$ (14972 characters), $hsbad$ (16877 characters), $hsascc2$ (51655 characters) and $hsasz1$ (66302 characters). The running times obtained for each sequence in A when we set $k \in \{30, 300, 600\}$ is shown in Table 1. We noticed that *KHD2* was the fastest implementation during the tests. This behavior suggests that the use of the Hamming distance to solve SSP-HD is probably a good choice in terms of time when using Tara sequence samples.

sequence in $A \setminus$ implementation	KHD1	KHD2	KHD3	KED
$hsarhgdig$	00:01:10	00:00:36	00:02:07	00:00:40
$hsankr10$	00:07:20	00:02:10	00:04:00	00:04:40
$hsaff4$	00:14:20	00:04:17	00:06:30	00:10:40
sequence in $A \setminus$ implementation	KHD1	KHD2	KHD3	KED
$hsarhgdig$	00:08:03	00:02:18	00:07:30	00:04:06
$hsankr10$	01:06:00	00:18:30	00:28:20	00:36:17
$hsaff4$	02:20:00	00:28:44	00:35:00	01:26:00
sequence in $A \setminus$ implementation	KHD1	KHD2	KHD3	KED
$hsarhgdig$	00:15:44	00:03:42	00:18:40	00:06:41
$hsankr10$	02:18:50	00:33:50	00:55:00	01:11:00
$hsaff4$	03:32:06	01:09:06	02:01:00	02:33:01

Table 1. Average running time (hours:minutes:seconds) obtained with the tests for sequences in A when they are compared to each sequence in B using $k = 30, 300$ and 600 respectively.

3.2. Tests with Tara sequences

From now on, we will use *KHD2* (because it has the best time for Hamming distance) and *KED* in the tests. To run the tests with Tara sequences, two samples were used: ERR599040 and ERR59914. Each sample has thousands of sequences. In order to estimate the time we would spend for comparing the two samples, we did some tests considering only 1, 10 and 100 sequences from ERR599040 (set A) and 2 million sequences from ERR599148 (set B).

Number of sequences considered in ERR599040	KHD2	KED
1	00:10:38	00:10:03
10	01:34:00	01:35:32
100	12:40:00	12:43:00

Table 2. Running time (hours:minutes:seconds) for run sequences in ERR599040 compared to 2,000,000 of sequences in ERR599148, with $k = 30$.

The running times obtained for the two restricted samples of the Tara project database are shown on Table 2. We noticed that *KHD2* and *KED* programs have similar run-times. Moreover, we estimate that it would take a long time to run the entire

sample set. This behavior justifies the reduction of sample sequences with some tool. It will be better discussed in Section 4.

4. Discussion, perspectives and conclusions

This work addresses the problem of specific strings (SSP). Two versions for previously described algorithms for the SSP-HD problem are showed with the second one presenting two variations. We implemented this versions and carried out tests with real *Homo sapiens* sequences data. We also performed a comparison with SSP-ED algorithm implemented by Dobre [Dobre 2017]. According to Table 1, the *KHD2* implementation has the better running time, suggesting that Hamming distance is indeed a promising measure for the considered problem at least concerning time consumption.

However, as well as it can be observed in Table 2, *KHD2* and *KED* show a slight difference in processing time, but both implementations are not fast enough for our main problem. Since we spend a reasonable time to compare a few sequences in ERR599040 with only thousands of sequences in ERR599148, we estimate that it would take hundreds of years to compare all the sequences. In order to overcome this difficult, we need to have a way to reduce the amount of sequences to be processed without giving up our central target that is to find sequences in one set that is not in another. Thus, we plan including a clustering preprocessing step with the CD-HIT tool [Li and Godzik 2006], more specifically the CD-HIT-2D, which is a tool that compares two protein databases and identifies similar sequences considering certain threshold. The FASTA file of non-similar sequences after clustering seems very interesting because it has unique sequences that can be tested using our algorithm. All other sequences can be discarded in this process. The CD-HIT used is [Fu et al. 2012].

Qty. of sequences	Time	Time (8 CPUs)	Clustered	ERR599148 Reduction
10000	00:01:06	00:00:12	3859	38.59%
20000	00:03:11	00:00:44	9957	49.78%
30000	00:08:20	00:01:32	17315	57.71%
40000	00:12:00	00:02:34	25465	63.66%
Qty. of sequences	Time	Time (8 CPUs)	Clustered	ERR599148 Reduction
10000	00:01:19	00:00:10	6360	63.60%
20000	00:04:23	00:00:33	15343	76.72%
30000	00:08:44	00:01:06	25060	85.54%
40000	00:14:16	00:01:48	35269	88.17%

Table 3. CD-HIT-2D Running time (hours:minutes:seconds) to 60% and 58% of similarity, with the same amount of sequences for ERR599040 and ERR599148.

Tests with TARA datasets are used to evaluate the behavior of the two sequential algorithms when it needs to handle a very large amount of data. It can be observed in Table 2 that both algorithms are not fast enough. The runtimes and percentage reductions of the number of sequences in ERR599148 are shown in Table 3. We noticed a significant reduction with 58% of the similarity that would lead us to conclude it is possible to find a good clustering, focusing on reducing the amount of sequences. In addition, we estimate that it takes about one week to cluster the dataset A and B with around 120

million of sequences. In the worst case, if $|A| = N$, $|B| = M$ and $KDH2 = O(n^2)$, it spends $O(NM \cdot n^2)$ for running all sequences. We plan to investigate new ways to cluster sequences to get smaller sets of sequences based on the CD-HIT. In addition, we want to search other tools or techniques for clustering that are faster than CD-HIT. Furthermore, since reducing the sample may not be enough, it is always necessary to improve the *KHD2* run-time. Moreano *et. al* [Feuser and Moreano 2018] describe a parallel approach for the *KED* algorithm achieving good speedups. Hence, we are considering a parallel computing approach for *KHD2* algorithm.

References

Bork, P., Bowler, C., de Vargas, C., Gorsky, G., Karsenti, E., and Wincker, P. (2015). Tara oceans studies plankton at planetary scale. *Science*, 348(6237):873–873.

Dobre, J. A. (2017). O problema da seleção de segmentos específicos: algoritmos e aplicações. Universidade Federal de Mato Grosso do Sul. Master's thesis.

Feuser, L. and Moreano, N. (2018). Parallel solutions to the k-difference primer problem. In *International Conference on Computational Science*, pages 506–523. Springer.

Fu, L., Niu, B., Zhu, Z., Wu, S., and Li, W. (2012). Cd-hit: accelerated for clustering the next-generation sequencing data. *Bioinformatics*, 28(23):3150–3152.

Gusfield, D. (1997). Algorithms on strings, trees and sequences: Computer science and computational biology, 1st editio. *New York, United States*, page 534.

Lanctot, J. K., Li, M., Ma, B., Wang, S., and Zhang, L. (2003). Distinguishing string selection problems. *Information and Computation*, 185(1):41–55.

Li, W. and Godzik, A. (2006). Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences. *Bioinformatics*, 22(13):1658–1659.

Maaß, M. G. (2003). A fast algorithm for the inexact characteristic string problem. Technical Report TUM-I0312, Fakultät für Informatik, TU München.

Montera, L. and Nicoletti, M. C. (2008). The PCR primer design as a metaheuristic search process. In *International Conference on Artificial Intelligence and Soft Computing*, pages 963–973. Springer.

Pesant, S., Not, F., Picheral, M., Kandels-Lewis, S., Le Bescot, N., Gorsky, G., Iudicone, D., Karsenti, E., Speich, S., Troublé, R., et al. (2015). Open science resources for the discovery and analysis of tara oceans data. *Scientific data*, 2:150023.

Shigefuku, R., Watanabe, T., Kanno, Y., Ikeda, H., Nakano, H., Hattori, N., Matsunaga, K., Matsumoto, N., Kanno, S.-i., Nosho, K., et al. (2017). Fusobacterium nucleatum detected simultaneously in a pyogenic liver abscess and advanced sigmoid colon cancer. *Anaerobe*, 48:144–146.

Zitvogel, L., Ma, Y., Raoult, D., Kroemer, G., and Gajewski, T. F. (2018). The microbiome in cancer immunotherapy: Diagnostic tools and therapeutic strategies. *Science*, 359(6382):1366–1370.