Computing Matching Statistics on Repetitive Texts

Younan Gao
Faculty of Computer Science
Dalhousie University
Halifax, Canada
yn803382@dal.ca

***Abstract*—Computing the *matching statistics* of a string $P[1..m]$ with respect to a text $T[1..n]$ is a fundamental problem which has application to genome sequence comparison. δ, as a relevant compressibility measure for repetitive texts, was recently introduced by Kociumaka et al. [1]. And they also proved that δ is an even smaller measure than γ, the smallest *string attractor* [2]. In this paper[1], we study the problem of computing the matching statistics upon highly repetitive texts. We present that within $O(\delta \lg \frac{n}{\delta})$ words of space, matching statistics can be computed in $O(m^2 \lg^{\epsilon} \gamma + m \lg n)$ time, where ϵ is an arbitrarily small positive constant.**

***Index Terms*—matching statistics, string attractor, repetitive texts**

I. Introduction

Matching statistics (MS) were introduced by Chang and Lawler [3] to solve the approximate string matching problem. The matching statistics MS of a pattern $P[1..m]$ with respect to a text $T[1..n]$ is an array storing a sequence of m integers such that MS$[i]$ is the length of the longest prefix of $P[i..m]$ that occurs in T. For example, given that T =“$aaabbbcc$” and P =“ccabb”, the matching statistics MS of P w.r.t. T, denoted by $MS_T(P)$ (or $MS(P)$, when the context is clear), stores an array of 5 integers, $\{2, 1, 3, 2, 1\}$. In addition to approximate string matching, the matching statistics also plays an important role in Bioinformatics. The genome sequence collections in many cases are highly repetitive. However, the compressed indexes based on statistical entropy might not capture repetitiveness [4]. In this paper, we focus on developing a new data structure to compute matching statistics that works for highly repetitive texts.

A. Related Work

A textbook solution [5] shows that a *suffix tree* data structure augmented with *suffix links* on the tree nodes can support to compute matching statistics in $O(m \lg \sigma)$ time, where σ represents the size of the universe that T is draws from. The data structure uses $O(n)$ words of space. Enno et al. [6] considered this problem on a fully compressed text indexes built upon T. Their indexes consist of a *wavelet tree* data structure that supports the LF-Mapping and the backward search, a LCP-array, and a data structure that supports fast-navigating on a *LCP-interval tree*. The indexes take $n \lg \sigma + 4n + o(n \lg \sigma)$ bits of space which allows to computer MS$_T(P)$ in $O(m \lg \sigma)$ time. Bannai et al. [7] considered to compute MS for a highly repetitive text. They augmented a *run-length BWT* with $O(r)$ words of space, where r is the number of *runs* in the BWT for T, to support computing MS in $O(m \lg \lg n)$ time, assuming that each element in T can be accessed in $O(\lg \lg n)$ time. Let z denote the number of phrases in the *Lempel-Ziv parsing*. It has been proved that $r = O(z \lg^2 n)$ holds for every text T [8].

Recently, the new compressibility measures γ, the smallest *string attractor*, and δ have been proposed. More details about γ and δ will be presented in the coming sections. Both new measures better capture the compressibility of repetitive strings. It has been proved that $\delta \leq \gamma \leq z = O(\delta \lg \frac{n}{\delta})$ [1], [2]. We will design the data structure whose space cost is measured by γ and δ for computing MS. We believe that our solutions outperform the current state-of-the-art algorithms in terms of the upper bound of the space cost.

B. Notation

Before introducing matching statistics, we give the precise definitions of the compressibility measures γ and δ that are mentioned above.

Definition 1. *[2] A string attractor of a string $T[1..n]$ is a set of γ' positions $\Gamma' = \{j_1, \cdots, j_{\gamma'}\}$ such that every substring $T[i..j]$ has an occurrence $T[i'..j'] = T[i..j]$ with $j_k \in [i', j']$ for some $j_k \in \Gamma'$.*

Let Γ^* denote $\{1, \Gamma, n\}$, where Γ denotes the smallest attractor sorted in an increasing order. For each $2 \leq i \leq |\Gamma^*|$, we call $T[\Gamma^*[i-1], \Gamma^*[i]]$ a *parsing phrase*. We use γ to denote the size of Γ. Given any substring $T[i..j]$, there must be an occurrence $T[i'..j'] = T[i..j]$ such that $T[i'..j']$ crosses the phrase boundary.

Kociumaka et al. [1] defined a new measure δ, which is even smaller than γ. Furthermore, δ can be computed in linear time.

Definition 2. *[1] Let $d_k(S)$ be the number of distinct length-k substrings in S. Then*

$$\delta = \max\{d_k(S)/k : k \in [1..n]\}.$$

The precise definition of matching statistics is as shown below.

Definition 3. *The matching statistics MS of a pattern $P[1..m]$ with respect to a text $T[1..n]$ are an array of integers $MS[1..m]$ such that $P[i..i+MS[i]-1]$ is the longest substring*

[1]This work is part of a course project for CSCI 6905 - Compact Data Structures in Computational Genomics, instructed by Prof. Travis Gagie.

of P starting at position i that matches substring somewhere in T.

Let T_1 and T_2 be two trees on the same set of leaves. We say a node from T_1 and a node from T_2 are *induced* together if they have a common leaf descendant. The *partner* operation originally from [9] is defined upon the inducing relationship. In [9], the operation is used to find the longest common substring. It is also an elementary operation in our solution. The definition of the partner operation is as shown below:

Definition 4. *[9] Given a pair of trees T_1 and T_2, the partner of a node $x \in T_1$ w.r.t a node $y \in T_2$, denoted by $partner(x/y)$, is the lowest ancestor y' of y, such that x and y' are induced. Likewise, $partner(y/x)$ is the lowest ancestor x' of x, such that x' and y are induced.*

Throughout this paper, we denote ϵ to be any small positive constant, and we study all problems in the standard *word RAM* model.

II. Preliminaries

In this section, we describe the previous results used in our solutions.

Lemma 1. *[10] Given n points in 2 dimensional rank space, there is a data structure using $f(n)$ words of space to support two-dimensional orthogonal range emptiness, reporting, and predecessor/successor queries in $g(n)$, $O((\mathsf{occ} + 1) \cdot g(n))$ and $O(g(n))$ time, respectively, so that,*

- *a) If $f(n) = O(n)$, then $g(n) = O(\lg^{\epsilon} n)$ query time.*
- *b) If $f(n) = O(n \lg\lg n)$, then $g(n) = O(\lg\lg n)$;*

Lemma 2. *[1] Given a string $S[1..n]$ with measure δ, there exists a data structure of size $O(\delta \lg \frac{n}{\delta})$ that can be built in $O(n \lg n)$ expected time and (1) can retrieve any substring $S[i..i+\ell]$ in time $O(\ell + \lg n)$, (2) can compute the Karp–Rabin fingerprint of any substring of S in time $O(\lg n)$, and (3) can report the occ occurrences of any pattern $P[1..m]$ in S in time $O(m \lg n + \mathsf{occ} \lg^{\epsilon} n)$.*

Lemma 3. *[9] (Induced-Check). Given two nodes x, y, where $x \in T_1$ and $y \in T_2$, we can check if they are induced or not in $O(\lg\lg n)$ time using an $O(n \lg\lg n)$ space structure, or in $O(\lg^{\epsilon} n)$ time using an $O(n)$ space structure.*

Lemma 4. *[9] (Find Partner). Given two nodes x, y where $x \in T_1$ and $y \in T_2$, we can find $partner(x/y)$ as well as $partner(y/x)$ in $O(\lg\lg n)$ time using an $O(n \lg\lg n)$ space structure, or in $O(\lg^{\epsilon} n)$ time using an $O(n)$ space structure.*

III. Computing MS within $O(\delta \lg \frac{n}{\delta})$ Words of Space

In this section, we describe the most space-efficient solution for computing MS.

A. Data Structures

First, we discuss the data structure part. Given a text $T[1..n]$, we build the following data structures:

- The smallest string attractor of T in $O(\gamma)$ words and $O(\gamma)$ parsing phrases defined upon the positions in the string attractor;
- The data structure with $O(\delta \lg \frac{n}{\delta})$ words of space shown in Lemma 2 for substring queries in T;
- One Patricia tree T_1 for the reversed parsing phrases, and another T_2 for the suffixes starting at phrase boundaries; Both require $O(\gamma)$ words of space;
- A linear space data structure for 4-sided emptiness (as induced-check), and range predecessor/successor (as *partner* searching) queries for the grid on which there is a marker at point (x, y) if the x-th phrase in right-to-left lexicographic order is followed in the parse by the lexicographically y-th suffix starting at a phrase boundary; All uses $O(\gamma)$ words of space;

Overall, the data structure requires $O(\delta \lg \frac{n}{\delta})$ words of space, since $\gamma = O(\delta \lg \frac{n}{\delta})$ [1].

B. The Algorithm for Computing MS

With the data structures introduced above, we can compute MS for a query pattern $P[1..m]$. As the pattern has m entries, P can be split into $m - 1$ different prefix and suffix pairs, namely $P[1..i]$ and $P[i+1..m]$ for each $1 \leq i \leq m-1$. Give a pair of such prefix and suffix, we need to find the loci of $(P[1..i])^{rev}$ (where the superscript rev indicates that a string is reversed) in T_1 and the loci of $P[i+1..m]$ in T_2, respectively, which can be solved by the following Lemma:

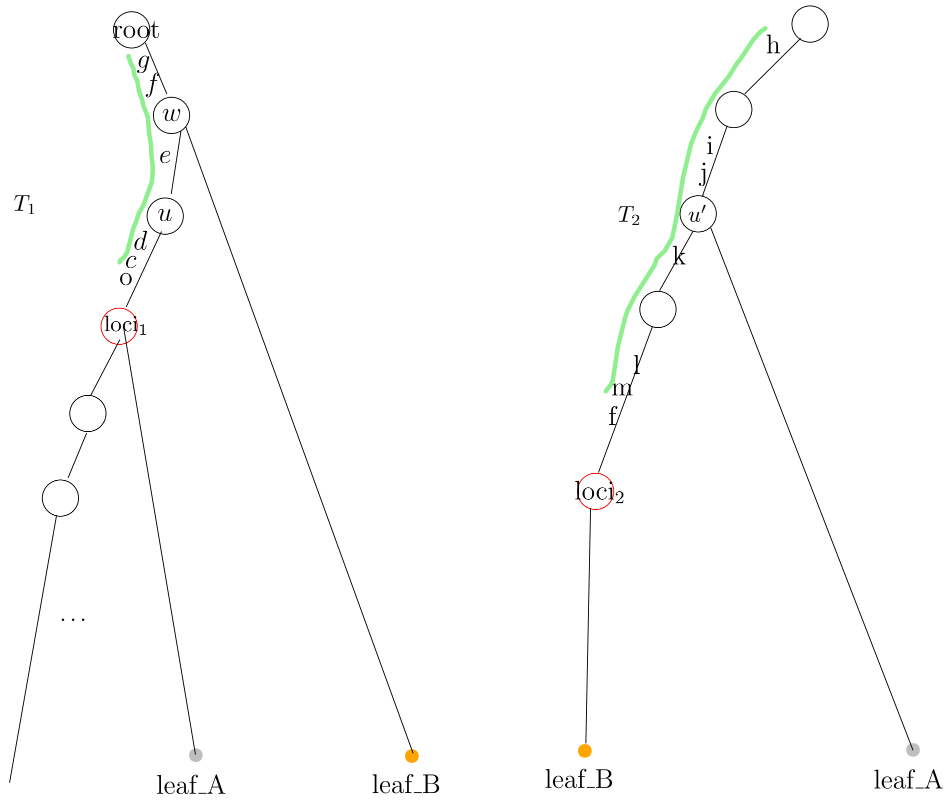
Lemma 5. *Given a pattern $P[1..m]$, for all $1 \leq i \leq m-1$, we can find the longest common prefix (LCP) of $(P[1..i])^{rev}$ and the path label of the node where the search in T_1 terminates, and the LCP of $P[i+1..m]$ and the path label of the node where the search in T_2 terminates in $O(m^2 + m \lg n)$ time.*

Proof. For each $1 \leq i \leq m-1$, a query need to access T to check that the path labels of the nodes where the searches terminate really are prefixed by some prefixes of $(P[1..i])^{rev}$ and $P[i+1..m]$, which can be solved by the substring queries using Lemma 2. As there are $m-1$ different pairs of $(P[1..i])^{rev}$ and $P[i+1..m]$ and the total number of characters that each pair of them contain is m, the searching time is $O(m^2 + m \lg n)$. $\square$

Next, we give the query algorithm in Theorem 1.

Theorem 1. *We can build a data structure for $T[0..n-1]$ with $O(\delta \lg \frac{n}{\delta})$ words of space, such that later, given $P[1..m]$, we can compute MS for P with respect to T in $O(m^2 \lg^{\epsilon} \gamma + m \lg n)$ time.*

Proof. For $1 \leq i \leq m-1$, we search for $(P[1..i])^{rev}$ in T_1 and for $P[i+1..m]$ in T_2; access T to find the longest common prefix (LCP) of $(P[1..i])^{rev}$ and the path label of the node where the search in T_{rev} terminates, and the LCP of $P[i+1..m]$ and the path label of the node where the search in T_{suf} terminates; take $loci_1$ and $loci_2$ to be the loci of those LCPs.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Pattern	a	b	c	d	e	f	g	h	i	j	k	l	m	n
MS			**8**	7	**6**	**8**	7							

Fig. 1. An example of computing MS using Theorem 1. Let $P[1..14] = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n\}$ denote the query pattern. The loci $loci_1$ and $loci_2$ of query string $P[1..7]$ and $P[8..14]$ is as shown in T_1 and T_2, respectively. The query algorithm iterates from $loci_1$ to the root node of T_1. Because $loci_1$ in T_1 and u' in T_2 are induced together, we set $MS[3]$ to 8. Next, as u in T_1 and u' in T_2 has the same leaf descendant, $MS[5]$ is set to 6. Finally, the last node we come across with before reaching the root node is w, which is induced with $loci_2$. Therefore, $MS[6]$ is set to 8.

Given a pair of locus $loci_1$ and $loci_2$, we iterate from the parent node of $loci_1$ to the root of T_1. At each node v visited, we retrieve the node u in T_2 such that $u = partner(v/loci_2)$. As defined before, u is the lowest ancestor of $loci_2$ that is induced together with v. We can see that $str(v).str(u)$ might be the longest substring of P starting at the position $(i - len(str(v)) + 1)$ that matches substring somewhere in T, where $str(v)$ (resp. $str(u)$) denotes the path labels of v (resp. u), and thus $MS[i - len(str(v)) + 1]$ could be $len(str(v).str(u))$. Note that, specially, if v (resp. u) is the loci, then $str(v)$ (resp. $str(u)$) denotes the longest matched prefix of $(P[1..i])^{rev}$ (resp. $P[i + 1..m]$). Given a position $k \in [1..m - 1]$, the longest substring of P starting at k can appear crossing $m-k+1$ different types of phrase boundaries, (i.e. the boundaries whose immediately left phrase ends with $P[j]$ and immediately right phrase starts with $P[j+1]$ for each $k \le j \le m-1$). It follows that $MS[i]$ is always the maximum length of those different types of longest substrings, because whenever a longer prefix of $P[i..m]$ that crosses some phrase boundary and matches somewhere in $T[1..n]$ is found, $MS[i]$ would be updated. The algorithm is shown in Algorithm 1. Note that there might be such cases that P contains some characters that do not appear in the alphabet universe that T is drawn from. The entries in MS corresponding to those unseen characters should be set to 0. To avoid missing those 0-entries in MS, initially, we set all entries in MS to 0.

We analyze the query time of the algorithm. As shown in Lemma 5, all locus of LCPs can be found in $O(m^2 + m \lg n)$ time. For each $1 \le i \le m - 1$, the while loop block can be operated $O(i)$ times. And all $O(i)$ iterations would call totally $O(i)$ times of $partner$-finding queries and fill at most i entries in MS. If only $O(\gamma)$ words of space is allowed, each $partner$-finding query requires $O(\lg^{\epsilon} \gamma)$ time as shown in Lemma 4. Therefore, the overall query time is $O(m^2 + m \lg n + \sum_{i=1}^{m-1} O(i \cdot \lg^{\epsilon} \gamma + i)) = O(m^2 \lg^{\epsilon} \gamma + m \lg n)$ time. $\square$

Algorithm 1 ComputingMS($P[1..m]$, T_{suff}, T_{ref})

```
1:  $MS[1..m] = \{0 \cdots 0\}$
2:  for $i = 1, 2, \ldots, m-1$ do
3:      Find $loci_1$ of $(P[1..i])^{Rev}$ in $T_1$
4:      Find $loci_2$ of $(P[i+1..m])$ in $T_2$
5:      $v \leftarrow parent(loci_1)$
6:      while v is not the root of $T_1$ do
7:          $u \leftarrow partner(v/loci_2)$
8:          $vp \leftarrow parent(v)$
9:          for $j = len(str(v))$, $j \geq len(str(vp))$, $j--$ do
10:             $\ell \leftarrow j + len(str(u))$
11:             if $\ell > MS[i-j+1]$ then
12:                 $MS[i-j+1] \leftarrow \ell$
13:         $v \leftarrow vp$
14: end
```

ACKNOWLEDGMENT

The author would like to thank Prof. Travis Gagie for sharing this interesting research topic in the course CSCI 6905.

REFERENCES

[1] T. Kociumaka, G. Navarro, and N. Prezza, "Towards a definitive measure of repetitiveness," in *Latin American Symposium on Theoretical Informatics*. Springer, 2021, pp. 207–219.

[2] D. Kempa and N. Prezza, "At the roots of dictionary compression: string attractors," in *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, 2018, pp. 827–840.

[3] W. I. Chang and E. L. Lawler, "Sublinear approximate string matching and biological applications," *Algorithmica*, vol. 12, no. 4, pp. 327–344, 1994.

[4] G. Navarro and V. Mäkinen, "Compressed full-text indexes," *ACM Computing Surveys (CSUR)*, vol. 39, no. 1, pp. 2–es, 2007.

[5] D. Gusfield, "Algorithms on stings, trees, and sequences: Computer science and computational biology," *Acm Sigact News*, vol. 28, no. 4, pp. 41–60, 1997.

[6] E. Ohlebusch, S. Gog, and A. Kügel, "Computing matching statistics and maximal exact matches on compressed full-text indexes," in *International Symposium on String Processing and Information Retrieval*. Springer, 2010, pp. 347–358.

[7] H. Bannai, T. Gagie, and I. Tomohiro, "Refining the r-index," *Theoretical Computer Science*, vol. 812, pp. 96–108, 2020.

[8] D. Kempa and T. Kociumaka, "Resolution of the burrows-wheeler transform conjecture," in *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2020, pp. 1002–1013.

[9] P. Abedin, S. Hooshmand, A. Ganguly, and S. V. Thankachan, "The heaviest induced ancestors problem revisited," in *Annual Symposium on Combinatorial Pattern Matching (CPM 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[10] T. M. Chan, K. G. Larsen, and M. Pătraşcu, "Orthogonal range searching on the ram, revisited," in *Proceedings of the twenty-seventh annual symposium on Computational geometry*, 2011, pp. 1–10.

[11] D. Harel and R. E. Tarjan, "Fast algorithms for finding nearest common ancestors," *siam Journal on Computing*, vol. 13, no. 2, pp. 338–355, 1984.