Fundamenta Informaticae XXI (2001) 1–7
DOI 10.3233/FI-2016-0000
IOS Press

On Computing Average Common Substring Over Run Length Encoded Sequences

Sahar Hooshmand
Department of Computer Science
University of Central Florida, Orlando, USA
sahar@cs.ucf.edu

Neda Tavakoli
School of Computational Science & Engineering
Georgia Institute of Technology, Atlanta, USA
neda.tavakoli@gatech.edu

Paniz Abedin
Department of Computer Science
University of Central Florida, Orlando, USA
paniz@cs.ucf.edu

Sharma V. Thankachan
Department of Computer Science
University of Central Florida, Orlando, USA
sharma.thankachan@ucf.edu

Abstract. The Average Common Substring (ACS) is a popular alignment-free distance measure for phylogeny reconstruction. The ACS of a sequence $\mathsf{X}[1, x]$ w.r.t. another sequence $\mathsf{Y}[1, y]$ is

$$\mathsf{ACS}(\mathsf{X}, \mathsf{Y}) = \frac{1}{x} \sum_{i=1}^{x} \max_{j} \mathsf{lcp}(\mathsf{X}[i, x], \mathsf{Y}[j, y])$$

Address for correspondence: Department of Computer Science, University of Central Florida, 117 Harris Center (Building 116), 4000 Central Florida Blvd., Orlando, Florida 32816-2362 USA. Phone: +1-407-823-5316.

The $\mathsf{lcp}(\cdot,\cdot)$ of two input sequences is the length of their longest common prefix. The ACS can be computed in $O(n)$ space and time, where $n = x + y$ is the input size. The compressed string matching is the study of string matching problems with the following twist: the input data is in a compressed format and the underling task must be performed with little or no decompression. In this paper, we revisit the ACS problem under this paradigm where the input sequences are given in their run-length encoded format. We present an algorithm to compute $\mathsf{ACS}(\mathsf{X},\mathsf{Y})$ in $O(N \log N)$ time using $O(N)$ space, where N is the total length of sequences after run-length encoding.

Keywords: String Algorithms, Suffix Trees, RL Encoding, Compression.

1. Introduction and Related Work

The Average Common Substring (ACS), proposed by Burstein *et al.* [1], is a simple alignment-free sequence comparison method. This measure and its various extensions [2, 3, 4, 5, 6, 7, 8, 9] have proven to be useful in multiple applications [10, 11, 12, 13, 14, 15]. Formally, ACS of a sequence $\mathsf{X}[1,x]$ w.r.t. another sequence $\mathsf{Y}[1,y]$, denoted by $\mathsf{ACS}(\mathsf{X},\mathsf{Y})$, is

$$\mathsf{ACS}(\mathsf{X},\mathsf{Y}) = \frac{1}{x}\sum_{i=1}^{x} L[i], \text{ where } L[i] = \max_{j} \mathsf{lcp}(\mathsf{X}[i,x],\mathsf{Y}[j,y])$$

The $\mathsf{lcp}(\cdot,\cdot)$ of two input sequences is the length of their longest common prefix. The (symmetric) distance based on ACS is [1]:

$$\mathsf{Dist}(\mathsf{X},\mathsf{Y}) = \frac{1}{2}\left(\frac{\log|\mathsf{Y}|}{\mathsf{ACS}(\mathsf{X},\mathsf{Y})} + \frac{\log|\mathsf{X}|}{\mathsf{ACS}(\mathsf{Y},\mathsf{X})}\right) - \frac{1}{2}\left(\frac{\log|\mathsf{X}|}{\mathsf{ACS}(\mathsf{X},\mathsf{X})} + \frac{\log|\mathsf{Y}|}{\mathsf{ACS}(\mathsf{Y},\mathsf{Y})}\right)$$

The computation of ACS is straightforward in $O(n)$ space and time using the generalized suffix tree of X and Y, where $n = x+y$ is the input size [1]. In this paper, we study the problem of computing ACS, where the input sequences are $\mathsf{X}'[1,x']$ and $\mathsf{Y}'[1,y']$, where $\mathsf{X}'[1,x']$ (resp., $\mathsf{Y}'[1,y']$) is the sequence corresponding to the run-length encoding of $\mathsf{X}[1,x]$ (resp., $\mathsf{Y}[1,y]$). Run-length encoding is a simple algorithm used for data compression in which runs of data (occurring of the same character on consecutive positions) are stored as a single character followed by the count of its consecutive occurrences. The challenge here is to design an algorithm for computing ACS in space and time close to $O(N)$ instead of $O(n)$, where $N = x' + y'$. We answer this question positively by presenting the following theorem.

Theorem 1.1. *Given two input sequences in their run-length encoded format, the distance between them based on the Average Common Substring (ACS) measure can be computed in $O(N)$ space and $O(N \log N)$ time, where N is the total length of sequences after run-length encoding.*

2. Notation and Background

Let Σ be the alphabet set from which the symbols in X and Y are drawn from. We denote X, X' and Y, Y' as follows:

$$\mathsf{X} = \alpha_1^{f_1}\alpha_2^{f_2}\alpha_3^{f_3}...\alpha_{x'}^{f_{x'}} \textbf{ and } \mathsf{X}' = (\alpha_1, f_1)(\alpha_2, f_2)(\alpha_3, f_3)...(\alpha_{x'}, f_{x'})$$

$$\mathsf{Y} = \beta_1^{g_1}\beta_2^{g_2}\beta_3^{g_3}...\beta_{y'}^{g_{y'}} \textbf{ and } \mathsf{Y}' = (\beta_1, g_1)(\beta_2, g_2)(\beta_3, g_3)...(\beta_{y'}, g_{y'})$$

Specifically, X is the concatenation of f_1 occurrences of α_1 followed by f_2 occurrences of α_2, and so on. Similarly, Y is the concatenation of g_1 occurrences of β_1 followed by g_2 occurrences of β_2, and so on. Moreover, $\alpha_i \neq \alpha_{i+1}$ and $\beta_i \neq \beta_{i+1}$ for all values of i. Here $\alpha_i, \beta_i \in \Sigma$ and $f_i, g_i \in \{1, 2, 3, ..., n\}$. The lexicographic order between two characters (c, k) and (c', k') in the encoded sequences is defined as follows: (c, k) is lexicographically smaller than (c', k') iff either c is lexicographically smaller than c' or $c = c'$ and $k < k'$. Also, define the suffixes

$$\mathsf{X}'[i, x'] = (\alpha_i, f_i)(\alpha_{i+1}, f_{i+1})...(\alpha_{x'}, f_{x'}) \textbf{ and}$$

$$\mathsf{Y}'[i, y'] = (\beta_i, g_i)(\beta_{i+1}, g_{i+1})...(\beta_{y'}, g_{y'})$$

If a suffix is a prefix of another suffix, we say that the shortest one is lexicographically smaller. Notice that for each $\mathsf{X}'[i, x']$ (resp., $\mathsf{Y}'[i, y']$), there exists an **equivalent suffix** $\mathsf{X}[F(i), x]$ (resp., $\mathsf{Y}[G(i), y]$) of X (resp., Y). Specifically,

$$F(i) = 1 + \sum_{k<i} f_k \textbf{ and } G(i) = 1 + \sum_{k<i} g_k$$

Observation 1. The kth lexicographically smallest suffix in $\mathcal{S}$ and the kth lexicographically smallest suffix in $\mathcal{S}'$ are equivalent for all values of $k \in [1, N]$, where

$$\mathcal{S} = \{\mathsf{X}[F(i), x] \mid 1 \leq i \leq x'\} \cup \{\mathsf{Y}[G(i), y] \mid 1 \leq i \leq y'\}$$

$$\textbf{and } \mathcal{S}' = \{\mathsf{X}'[i, x'] \mid 1 \leq i \leq x'\} \cup \{\mathsf{Y}'[i, y'] \mid 1 \leq i \leq y'\}$$

Example 1. Consider two input sequences $X = CCCCCAAAGG$ and $Y = CCAATTTGGGG$. According to the definition of X' and Y', $\mathsf{X}' = (C, 5)(A, 3)(G, 2)$ and $\mathsf{Y}' = (C, 2)(A, 2)(T, 3)(G, 4)$. For each suffix of $\mathsf{X}'[i, x]$ there exists an equivalent suffix $\mathsf{X}[F(i), x]$. For instance, $\mathsf{X}[F(2), 10] = \mathsf{X}[6, 10]$ is the equivalent suffix of $\mathsf{X}'[2, 3]$.

The main component of our algorithm is a trie $\mathcal{T}$ over all strings in $\mathcal{S}$. It consists of N leaves and at most $N - 1$ internal nodes. Each leaf node in $\mathcal{T}$ corresponds to a unique suffix in $\mathcal{S}$. Specifically, the ith leftmost leaf ℓ_i corresponds to the ith lexicographically smallest suffix in $\mathcal{S}$. Each internal node v is associated with two values, (i) $\mathsf{nodeDepth}(v)$: the number of nodes on the path from root to v and (ii) $\mathsf{strDepth}(v)$: the length of the longest common prefix over all suffixes corresponding to the leaves under v. Additionally, we call a leaf type-X (resp., type-Y) if the suffix corresponding to it is from X (resp., Y). The space occupancy of $\mathcal{T}$ is $O(N)$ words.

Lemma 2.1. The trie $\mathcal{T}$ can be constructed in $O(N \log N)$ time using $O(N)$ space.

Proof. We construct a generalized suffix tree of X' and Y' and then convert it into $\mathcal{T}$ [16, 17, 18] by exploring Observation 1.

3. An $O(N)$-Space and $O(n \log N)$-Time Algorithm

The first step is to construct $\mathcal{T}$ from X' and Y'. Then, we associate each leaf node (except two[1]) with two values, $\mathsf{char}(\cdot)$ and $\mathsf{freq}(\cdot)$ as follows: Let ℓ_a be the leaf corresponding to $\mathsf{X}[F(i+1), x]$. Then

$$\mathsf{char}(\ell_a) = \alpha_i \text{ and } \mathsf{freq}(\ell_a) = f_i$$

Similarly, let ℓ_b be the leaf corresponding to $\mathsf{Y}[G(j+1), y]$. Then

$$\mathsf{char}(\ell_b) = \beta_j \text{ and } \mathsf{freq}(\ell_b) = g_j$$

For each $\sigma \in \Sigma$, define (and compute)

$$\mathsf{maxRun}(\sigma) = \max\{g_k \mid k \in [1, y'] \textbf{ and } \beta_k = \sigma\}$$

The key intuition behind our algorithm is the following simple observation.

Observation 2. Let ℓ_a be the leaf in $\mathcal{T}$ corresponding to the suffix $\mathsf{X}[F(i+1), x]$ for an $i \in [1, x'-1]$. Also, let $\mathsf{X}[p, x] = \alpha_i^h \circ \mathsf{X}[F(i+1), x]$ for some $h \in [1, f_i]$. Specifically, $p = F(i+1) - h$ and "$\circ$" denotes concatenation. Then $L[p]$:

- is $\mathsf{maxRun}(\alpha_i)$ if $h > \mathsf{maxRun}(\alpha_i)$ and
- is $h + \mathsf{strDepth}(v)$ otherwise, where node v is the lowest ancestor of ℓ_a such that there exists a type-Y leaf under v with $\mathsf{char}(\cdot) = \alpha_i$ and $\mathsf{freq}(\cdot) \geq h$.

We now present an efficient algorithm for computing $L[\cdot]$'s based on the above observation. First we construct a collection $\{\mathcal{T}_\sigma \mid \sigma \in \Sigma\}$ of new tries from $\mathcal{T}$. Specifically, the $\mathcal{T}_\sigma$ is a compact trie over all those suffixes in $\mathcal{T}$, such that $\mathsf{char}(\cdot)$ of the leaves corresponding to them is σ. The total number of nodes over all $\mathcal{T}_\sigma$'s is $O(N)$ as each leaf node in $\mathcal{T}$ belongs to exactly one $\mathcal{T}_\sigma$. Moreover, they can be extracted from $\mathcal{T}$ in $O(N)$ total time.

Next, we pre-process each $\mathcal{T}_\sigma$ in time linear to its size for answering level ancestor queries in constant time [19]. A level ancestor query (v, l) asks to return the ancestor u of v with $\mathsf{nodeDepth}(u) = l$. Finally, for each internal node v in each $\mathcal{T}_\sigma$, we compute $\mathsf{freq}(v)$, which is the maximum over $\mathsf{freq}(\cdot)$'s of all type-Y leaves under v. Note that $\mathsf{freq}(v) = 0$ if all leaves under u are of type-X. This step can also be implemented in linear time via a bottom up traversal of $\mathcal{T}_\sigma$.

We are now ready to present the final steps of our algorithm. For $p = 1, 2, 3, ..., x$, we compute $L[p]$ using Algorithm 1 (see the pseudo-code below). Note that the node v can be computed via a binary search (using level ancestor queries) over the nodes on the path to root in $O(\log N)$ time. This completes the description of our algorithm. The correctness is immediate from Observation 2. The total time complexity is $O(n \log N)$.

[1] Specifically, the leaves corresponding to $\mathsf{X}[F(x'), x]$ and $\mathsf{Y}[G(y'), y]$.

Algorithm 1: Computing L[p]

Let $\mathsf{X}[p, x] = \alpha_i^h \circ \mathsf{X}[F(i+1), x]$
if $h > \mathsf{maxRun}(\alpha_i)$ **then** $L[p] = \mathsf{maxRun}(\alpha_i)$
else
Find the leaf node w in $\mathcal{T}_{\alpha_i}$ corresponding to $\mathsf{X}[F(i+1), x]$ and its lowest ancestor v such that $\mathsf{freq}(v) \geq h$
Fix v as the root when $i = x'$
$L[p] = h + \mathsf{strDepth}(v)$
endif

4. An $O(N)$-Space and $O(N \log N)$-Time Algorithm

Define $A[i]$ for $i = 1, 2, 3, ..., x'$, where

$$A[i] = \sum_{p=F(i)}^{F(i+1)-1} L[p]$$

Therefore,

$$\mathsf{ACS}(\mathsf{X}, \mathsf{Y}) = \frac{1}{x} \sum_{i=1}^{x'} A[i]$$

We now present a new algorithm in which we compute each $A[i]$ in $O(\log N)$ time. For each internal node v in $\mathcal{T}_\sigma$, define $\mathsf{weight}(v)$ as follows: $\mathsf{weight}(\cdot)$ of the root node is 0. For any other node v with v' being its parent,

$$\mathsf{weight}(v) = \mathsf{weight}(v') + \mathsf{freq}(v) \times (\mathsf{strDepth}(v) - \mathsf{strDepth}(v'))$$

By performing a top-down tree traversal, we compute $\mathsf{weight}(\cdot)$ over all internal nodes in $\mathcal{T}_\sigma$ in time linear to its size. Therefore, time over all $\mathcal{T}_\sigma$'s is $O(N)$. We now compute $A[i]$'s using the following steps.

- For any $i \in [1, x' - 1]$, we first find the leaf node w in $\mathcal{T}_{\alpha_i}$ corresponding to the suffix $\mathsf{X}[F(i+1), x]$. Also, find the lowest ancestor v of w, such that there exists a type-Y leaf under v (equivalently $\mathsf{freq}(v) \neq 0$) via binary search using level ancestors queries. This step takes $O(\log N)$ time. Next, we have two cases and we handle them separately as follows. For brevity, let $m = \mathsf{maxRun}(\alpha_i)$.

  - If $f_i > m$, then

    $$A[i] = \mathsf{weight}(v) + (1 + 2 + 3 + ... + m) + m(f_i - m)$$

    By simplifying, we have $A[i] = \mathsf{weight}(v) + m(f_i - (m-1)/2)$.

  - If $f_i \leq m$, then find the lowest ancestor u of w, such that $\mathsf{freq}(u) \geq f_i$ (via binary search using level ancestors queries). Then,

$$A[i] = \mathsf{weight}(v) - \mathsf{weight}(u) + f_i \times \mathsf{strDepth}(u) + f_i(1 + f_i)/2$$

- For $i = x'$, $A[i]$
  - is $(1 + 2 + ... + f_i) = f_i(f_i + 1)/2$ if $f_i \leq m$ and
  - is $(1 + 2 + ... + m) + m(f_i - m) = m(f_i - (m - 1)/2)$, otherwise.

In summary, the time complexity is $O(N \log N)$ plus $O(\log N)$ for each i in $[1, x']$. Therefore, total time is $O(N \log N)$. The correctness follows from Observation 2 and the definition of $\mathsf{weight}(\cdot)$. This completes the proof of Theorem 1.

Acknowledgments

This research is supported in part by the U.S. National Science Foundation under CCF-1704552 and CCF-1703489.

References

[1] Burstein D, Ulitsky I, Tuller T, Chor B. Information theoretic approaches to whole genome phylogenies. In: Proceedings of the 9th Annual International Conference on Research in Computational Molecular Biology (RECOMB). 2005 pp. 283–295.

[2] Aluru S, Apostolico A, Thankachan SV. Efficient alignment free sequence comparison with bounded mismatches. In: Proceedings of the 19th Annual International Conference on Research in Computational Molecular Biology (RECOMB). 2015 pp. 1–12.

[3] Apostolico A, Guerra C, Landau GM, Pizzi C. Sequence similarity measures based on bounded hamming distance. *Theoretical Computer Science*, 2016. **638**:76–90.

[4] Leimeister CA, Morgenstern B. *kmacs*: the k-mismatch average common substring approach to alignment-free sequence comparison. *Bioinformatics*, 2014. **30**(14):2000–2008.

[5] Manzini G. Longest Common Prefix with Mismatches. In: Proceedings of the 22nd International Symposium on String Processing and Information Retrieval (SPIRE). Springer, 2015 pp. 299–310.

[6] Thankachan SV, Apostolico A, Aluru S. A Provably Efficient Algorithm for the k-Mismatch Average Common Substring Problem. *Journal of Computational Biology*, 2016. **23**(6):472–482.

[7] Thankachan SV, Chockalingam SP, Liu Y, Apostolico A, Aluru S. ALFRED: a practical method for alignment-free distance computation. *Journal of Computational Biology*, 2016. **23**(6):452–460.

[8] Thankachan SV, Chockalingam SP, Liu Y, Krishnan A, Aluru S. A greedy alignment-free distance estimator for phylogenetic inference. In: Proceedings of 5th International Conference on Computational Advances in Bio and Medical Sciences (ICCABS). 2015 .

[9] Thankachan SV, Aluru C, Chockalingam SP, Aluru S. Algorithmic Framework for Approximate Matching Under Bounded Edits with Applications to Sequence Analysis. In: Proceedings of the 922nd Annual International Conference on Research in Computational Molecular Biology (RECOMB). 2018 pp. 211–224.

[10] Apostolico A. Maximal words in sequence comparisons based on subword composition. In: Algorithms and Applications, pp. 34–44. Springer, 2010.

[11] Bonham-Carter O, Steele J, Bastola D. Alignment-free genetic sequence comparisons: a review of recent approaches by word analysis. *Briefings in bioinformatics*, 2013. p. bbt052.

[12] Chang G, Wang T. Phylogenetic analysis of protein sequences based on distribution of length about common substring. *The Protein Journal*, 2011. **30**(3):167–172.

[13] Comin M, Verzotto D. Alignment-free phylogeny of whole genomes using underlying subwords. *Algorithms for Molecular Biology*, 2012. **7**(1):1.

[14] Domazet-Lošo M, Haubold B. Efficient estimation of pairwise distances between genomes. *Bioinformatics*, 2009. **25**(24):3221–3227.

[15] Guyon F, Brochier-Armanet C, Guénoche A. Comparison of alignment free string distances for complete genome phylogeny. *Advances in Data Analysis and Classification*, 2009. **3**(2):95–108.

[16] Weiner P. Linear pattern matching algorithms. In: Proceedings of the 14th Annual IEEE Symposium on Switching and Automata Theory (SWAT). 1973 pp. 1–11.

[17] McCreight EM. A space-economical suffix tree construction algorithm. *Journal of the ACM (JACM)*, 1976. **23**(2):262–272.

[18] Farach M. Optimal Suffix Tree Construction with Large Alphabets. In: 38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997. 1997 pp. 137–143.

[19] Bender MA, Farach-Colton M. The LCA Problem Revisited. In: LATIN 2000: Theoretical Informatics, 4th Latin American Symposium, Punta del Este, Uruguay, April 10-14, 2000, Proceedings. 2000 pp. 88–94.