Range Shortest Unique Substring Queries

Paniz Abedin[1(✉)], Arnab Ganguly[2], Solon P. Pissis[3], and Sharma V. Thankachan[1]

[1] Department of Computer Science, University of Central Florida, Orlando, USA
paniz@cs.ucf.edu, sharma.thankachan@ucf.edu
[2] Department of Computer Science, University of Wisconsin - Whitewater, Whitewater, USA
gangulya@uww.edu
[3] CWI, Amsterdam, The Netherlands
solon.pissis@cwi.nl

Abstract. Let $\mathsf{T}[1,n]$ be a string of length n and $\mathsf{T}[i,j]$ be the substring of T starting at position i and ending at position j. A substring $\mathsf{T}[i,j]$ of T is a repeat if it occurs more than once in T; otherwise, it is a unique substring of T. Repeats and unique substrings are of great interest in computational biology and in information retrieval. Given string T as input, the *Shortest Unique Substring* problem is to find a shortest substring of T that does not occur elsewhere in T. In this paper, we introduce the range variant of this problem, which we call the *Range Shortest Unique Substring* problem. The task is to construct a data structure over T answering the following type of online queries efficiently. Given a range $[\alpha,\beta]$, return a shortest substring $\mathsf{T}[i,j]$ of T with exactly one occurrence in $[\alpha,\beta]$. We present an $\mathcal{O}(n\log n)$-word data structure with $\mathcal{O}(\log_w n)$ query time, where $w=\Omega(\log n)$ is the word size. Our construction is based on a non-trivial reduction allowing us to apply a recently introduced optimal geometric data structure [Chan et al. ICALP 2018].

Keywords: Shortest unique substring · Suffix tree · Heavy-light decomposition · Range queries · Geometric data structures

1 Introduction

Finding regularities in strings is one of the main topics of combinatorial pattern matching and its applications. Among the most well-studied types of string regularities is the notion of repeat. Let $\mathsf{T}[1,n]$ be a string of length n. A substring $\mathsf{T}[i,j]$ of T is called a repeat if it occurs more than once in T. The notion of unique substring is thus dual: it is a substring $\mathsf{T}[i,j]$ of T that does not occur more than once in T. Computing repeats and unique substrings has applications in computational biology [14,23] and in information retrieval [19,22].

Supported in part by the U.S. National Science Foundation under CCF-1703489 and the Royal Society International Exchanges Scheme (IES\R1\180175).

N. R. Brisaboa and S. J. Puglisi (Eds.): SPIRE 2019, LNCS 11811, pp. 258–266, 2019.
https://doi.org/10.1007/978-3-030-32686-9_18

In this paper, we are interested in the notion of shortest unique substring. All shortest unique substrings of T can be computed in $\mathcal{O}(n)$ time using the suffix tree data structure [9,29]. Many different problems based on this notion have already been studied. Pei et al. [22] considered the following problem on the so-called position (or point) queries. Given a position i of T, return a shortest unique substring of T covering i. The authors gave an $\mathcal{O}(n^2)$-time and $\mathcal{O}(n)$-space algorithm, which finds the shortest unique substring covering every position of T. Since then, the problem has been revisited and optimal $\mathcal{O}(n)$-time algorithms have been presented by Ileri et al. [16] and by Tsuruta et al. [27]. Several other variants of this problem have been investigated [2,10,11,15,18,20,21,24,28].

We introduce a natural generalization of the shortest unique substring problem. Specifically, our focus is on the range version of the problem, which we call the *Range Shortest Unique Substring* (rSUS) problem. The task is to construct a data structure over T to be able to answer the following type of online queries efficiently. Given a range $[\alpha, \beta]$, return a shortest substring $\mathsf{T}[k, k+h-1]$ of T with exactly one occurrence in $[\alpha, \beta]$; i.e., $k \in [\alpha, \beta]$, there is no $k' \in [\alpha, \beta]$ such that $\mathsf{T}[k, k+h-1] = \mathsf{T}[k', k'+h-1]$, and h is minimal.

Range queries are a classic data structure topic [6,7,30]. A range query $q = f(A, i, j)$ on an array of n elements over some set S, denoted by $A[1, n]$, takes two indices $1 \leq i \leq j \leq n$, a function f defined over arrays of elements of S, and outputs $f(A[i, j]) = f(A[i], \ldots, A[j])$. Range query data structures have also been considered specifically for strings [1,3,4,12]. For instance, in bioinformatics applications we are often interested in finding regularities in certain regions of a DNA sequence [5,17]. In the Range-LCP problem, defined by Amir et al. [3], the task is to construct a data structure over T to be able to answer the following type of online queries efficiently. Given a range $[\alpha, \beta]$, return $i, j \in [\alpha, \beta]$ such that the length of the longest common prefix of $\mathsf{T}[i, n]$ and $\mathsf{T}[j, n]$ is maximal among all pairs of suffixes within this range. The state of the art is an $\mathcal{O}(n)$-word data structure supporting $\mathcal{O}(\log^{\mathcal{O}(1)} n)$-time queries [1] (see also [12]).

Main Problem and Main Result

An *alphabet* Σ is a finite nonempty set of elements called *letters*. We fix a *string* $\mathsf{T}[1, n] = \mathsf{T}[1] \cdots \mathsf{T}[n]$ over Σ. The *length* of T is denoted by $|\mathsf{T}| = n$. By $\mathsf{T}[i, j] = \mathsf{T}[i] \cdots \mathsf{T}[j]$, we denote the *substring* of T starting at position i and ending at position j of T. We say that another string P has an *occurrence* in T or, more simply, that P *occurs* in T if $\mathsf{P} = \mathsf{T}[i, i+|\mathsf{P}|-1]$, for some i. Thus, we characterize an occurrence of P by its starting position i in T. A *prefix* of T is a substring of T of the form $\mathsf{T}[1, i]$ and a *suffix* of T is a substring of T of the form $\mathsf{T}[i, n]$.

We next formally define the main problem considered in this paper.

Problem rSUS
Preprocess: String $\mathsf{T}[1, n]$.
Query: Range $[\alpha, \beta]$, where $1 \leq \alpha \leq \beta \leq n$.
Output: (p, ℓ) such that $\mathsf{T}[p, p+\ell-1]$ is a shortest string with exactly one occurrence in $[\alpha, \beta]$.

If $\alpha = \beta$ the answer $(\alpha, 1)$ is trivial. So, in the rest we assume that $\alpha < \beta$.

Example 1. Given $\mathsf{T} = \overset{1\,2\,3\,4\,5\,6\,7\,8\,9\,10\,11\,12\,13\,14\,15\,16\,17\,18\,19\,20\,21}{\textbf{caabcaddaacaddaaaabac}}$ and a query $[\alpha, \beta] = [5, 16]$, we need to find a shortest substring of T with exactly one occurrence in $[5, 16]$. The output here is $(p, \ell) = (10, 2)$, because $\mathsf{T}[10, 11] = \textbf{ac}$ is the shortest substring of T with exactly one occurrence in $[5, 16]$.

In what follows, we prove our main result (Theorem 1).

Theorem 1. *We can construct an $\mathcal{O}(n \log n)$-word data structure which answers* rSUS *queries in $\mathcal{O}(\log_w n)$ time per query in the word RAM model, where $w = \Omega(\log n)$ is the word size.*

Our construction is based on ingredients such as the suffix tree [29], heavy-light decomposition [25], and a geometric data structure for rectangle stabbing [8].

2 Our Data Structure

Let us start with some definitions.

Definition 1. *For a position $k \in [1, n]$ and $h \geq 1$, we define* $\mathsf{Prev}(k, h)$ *and* $\mathsf{Next}(k, h)$ *as follows:*

$$\mathsf{Prev}(k, h) = \max_j \{\{j < k \mid \mathsf{T}[k, k + h - 1] = \mathsf{T}[j, j + h - 1]\} \cup \{-\infty\}\}$$
$$\mathsf{Next}(k, h) = \min_j \{\{j > k \mid \mathsf{T}[k, k + h - 1] = \mathsf{T}[j, j + h - 1]\} \cup \{+\infty\}\}.$$

Intuitively, let x and y be the occurrences of $\mathsf{T}[k, k + h - 1]$ right before and right after the position k, respectively. Then, $\mathsf{Prev}(k, h) = x$ and $\mathsf{Next}(k, h) = y$. If x (resp., y) does not exist, then $\mathsf{Prev}(k, h) = -\infty$ (resp., $\mathsf{Next}(k, h) = +\infty$).

Definition 2. *Let $k \in [a, b]$. We define $\lambda(a, b, k)$ as follows:*

$$\lambda(a, b, k) = \min\{h \mid \mathsf{Prev}(k, h) < a \quad \textbf{\textit{and}} \quad \mathsf{Next}(k, h) > b\}.$$

Intuitively, $\lambda(a, b, k)$ denotes the length of the shortest substring that starts at position k with exactly one occurrence in $[a, b]$.

Definition 3. *For a position $k \in [1, n]$, we define C_k as follows:*

$$C_k = \{h \mid (\mathsf{Next}(k, h), \mathsf{Prev}(k, h)) \neq (\mathsf{Next}(k, h - 1), \mathsf{Prev}(k, h - 1))\}.$$

Example 2 (Running Example)**.** Let $\mathsf{T} = \overset{1\,2\,3\,4\,5\,6\,7\,8\,9\,10\,11\,12\,13\,14\,15\,16\,17\,18\,19\,20\,21}{\textbf{caabcaddaacaddaaaabac}}$ and $k = 10$. We have that $(\mathsf{Next}(10, 1), \mathsf{Prev}(10, 1)) = (12, 9)$, $(\mathsf{Next}(10, 2), \mathsf{Prev}(10, 2)) = (20, -\infty)$, and $(\mathsf{Next}(10, 3), \mathsf{Prev}(10, 3)) = (+\infty, -\infty)$. Thus, $C_{10} = \{2, 3\}$.

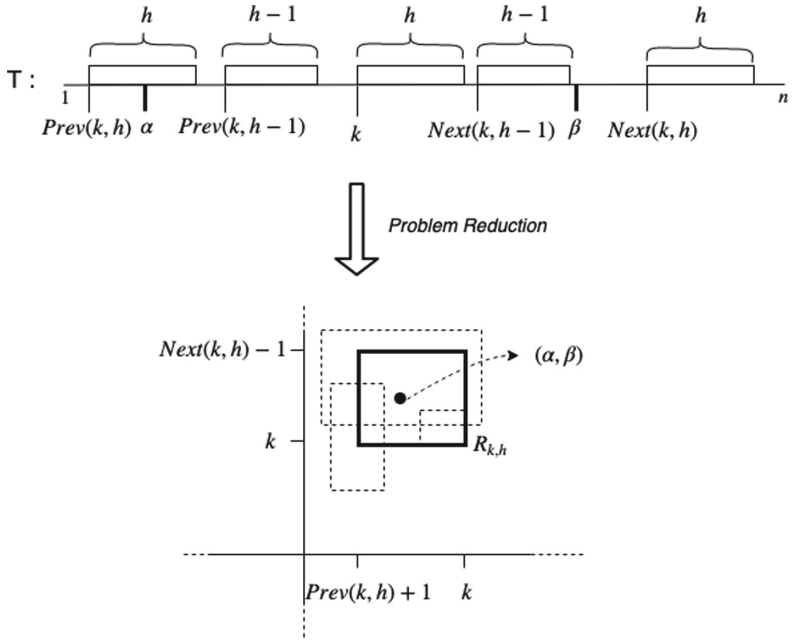

Fig. 1. Illustration of the problem reduction: (k, h) is the output of the rSUS problem with query range $[\alpha, \beta]$, where $h = \lambda(\alpha, \beta, k) \in C_k$. $R_{k,h}$ is the lowest weighted rectangle in $\mathcal{R}$ containing the point (α, β).

Intuitively, C_k stores the set of candidate lengths for shortest unique substrings starting at position k. We make the following observation.

Observation 1. *$\lambda(a, b, k) \in C_k$, for any $1 \leq a \leq b \leq n$.*

Example 3 (Running Example)**.** Let $\mathsf{T} = \overset{1\,2\,3\,4\,5\,6\,7\,8\,9\,10\,11\,12\,13\,14\,15\,16\,17\,18\,19\,20\,21}{\mathbf{caabcaddaacaddaaaabac}}$ and $k = 10$. We have that $C_{10} = \{2, 3\}$. For $a = 5$ and $b = 16$, $\lambda(5, 16, 10) = 2$, denoting substring **ac**. For $a = 5$ and $b = 20$, $\lambda(5, 20, 10) = 3$, denoting substring **aca**.

The following combinatorial lemma is crucial for efficiency.

Lemma 1. $\sum_k |C_k| = \mathcal{O}(n \log n)$.

The proof of Lemma 1 is deferred to Sect. 3.

We are now ready to present our construction. By Observation 1, for a given query range $[\alpha, \beta]$, the answer (p, ℓ) we are looking for is the pair (k, h) with the minimum h under the following conditions: $k \in [\alpha, \beta]$, $h \in C_k$, $\mathsf{Prev}(k, h) < \alpha$ and $\mathsf{Next}(k, h) > \beta$. Equivalently, (p, ℓ) is the pair (k, h) with the minimum h, such that $h \in C_k$, $\alpha \in (\mathsf{Prev}(k, h), k]$, and $\beta \in [k, \mathsf{Next}(k, h))$. We map each $h \in C_k$ into a weighted rectangle $R_{k,h}$ with weight h and defined as follows:

$$R_{k,h} = [\mathsf{Prev}(k, h) + 1, k] \times [k, \mathsf{Next}(k, h) - 1].$$

Let $\mathcal{R}$ be the set of all such rectangles, then the lowest weighted rectangle in $\mathcal{R}$ stabbed by the point (α, β) is $R_{p,\ell}$. In short, an rSUS query on $\mathsf{T}[1, n]$ with an input range $[\alpha, \beta]$ can be reduced to an equivalent top-1 rectangle stabbing query on a set $\mathcal{R}$ of rectangles with input point (α, β), where the task is to report the lowest weighted rectangle in $\mathcal{R}$ containing the point (α, β) (see Fig. 1 for an illustration). By Lemma 1, we have that $|\mathcal{R}| = \mathcal{O}(n \log n)$. Therefore, by employing the optimal data structure for top-1 rectangle stabbing presented by Chan et al. [8], which takes $\mathcal{O}(|\mathcal{R}|)$-word space supporting $\mathcal{O}(\log_w |\mathcal{R}|)$-time queries, we obtain the space-time trade-off in Theorem 1. This completes our construction.

3 Proof of Lemma 1

Let $\mathsf{lcp}(i, j)$ denote the length of the longest common prefix of the suffixes of T starting at positions i and j in T. Also, let S denote the set of all (x, y) pairs, such that $1 \leq x < y \leq n$ and $\mathsf{lcp}(x, y) > \mathsf{lcp}(x, z)$, for all $z \in [x + 1, y - 1]$.

The proof of Lemma 1 can be broken down into the following two lemmas.

Lemma 2. $\sum_k |C_k| = \mathcal{O}(|S|)$.

Lemma 3. $|S| = \mathcal{O}(n \log n)$.

3.1 Proof of Lemma 2

Let us fix a position k. Let

$$C'_k = \{h \mid \mathsf{Prev}(k, h) \neq \mathsf{Prev}(k, h - 1)\}$$
$$C''_k = \{h \mid \mathsf{Next}(k, h) \neq \mathsf{Next}(k, h - 1)\}.$$

Clearly we have that $C_k = C'_k \cup C''_k$.

The following statements can be deduced by a simple contradiction argument:

1. Let $i = \mathsf{Prev}(k, h) \neq -\infty$, where $h \in C'_k$, then $i = \mathsf{Prev}(k, \mathsf{lcp}(i, k))$
2. Let $j = \mathsf{Next}(k, h) \neq \infty$, where $h \in C''_k$, then $j = \mathsf{Next}(k, \mathsf{lcp}(k, j))$.

Figure 2 illustrates the proof for the first statement. The second one can be proved in a similar fashion.

Clearly, $|C'_k|$ is proportional to the number of (i, k) pairs such that $\mathsf{lcp}(i, k) \neq 0$ and $i = \mathsf{Prev}(k, \mathsf{lcp}(i, k))$. Similarly, $|C''_k|$ is proportional to the number of (k, j) pairs such that $\mathsf{lcp}(k, j) \neq 0$ and $j = \mathsf{Next}(k, \mathsf{lcp}(k, j))$. Therefore, $\sum_k |C_k|$ is proportional to the number of (x, y) pairs, such that $\mathsf{lcp}(x, y) \neq 0$ and $\mathsf{lcp}(x, y) > \mathsf{lcp}(x, z)$, for all $z \in [x + 1, y - 1]$. This completes the proof of Lemma 2.

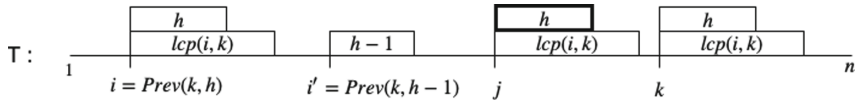

Fig. 2. Let $h \in C'_k$ and $i = \mathsf{Prev}(k, h)$. By contradiction, assume that there exists $j \in (i, k)$ such that $j = \mathsf{Prev}(k, \mathsf{lcp}(i, k))$. Since $h \leq \mathsf{lcp}(i, k)$, $\mathsf{T}[j, j+h-1] = \mathsf{T}[k, k+h-1]$. This is a contradiction with $i = \mathsf{Prev}(k, h)$. Thus, $i = \mathsf{Prev}(k, \mathsf{lcp}(i, k))$.

3.2 Proof of Lemma 3

Consider the suffix tree data structure of string $\mathsf{T}[1, n]$, which is a compact trie of the n suffixes of T appended with a letter $\$ \notin \Sigma$ [29]. This suffix tree consists of n leaves (one for each suffix of T) and at most $n-1$ internal nodes. The edges are labeled with substrings of T. Let u be the lowest common ancestor of the leaves corresponding to the strings $\mathsf{T}[x, n]\$$ and $\mathsf{T}[y, n]\$$. Then, the concatenation of the edge labels on the path from the root to u is exactly the longest common prefix of $\mathsf{T}[x, n]\$$ and $\mathsf{T}[y, n]\$$. For any node u, we denote by $\mathsf{size}(u)$ the total number of leaf nodes of the subtree rooted at u.

We decompose the nodes in the suffix tree into *light* and *heavy* nodes. The root node is light and for any internal node, exactly one child is heavy. Specifically, the heavy child is the one having the largest number of leaves in its subtree (ties are broken arbitrarily). All other children are light. This tree decomposition is known as *heavy-light decomposition*. We have the following critical observation. Any path from the root to a leaf node contains many nodes, however, the number of light nodes is at most $\log n$ [13,25]. We have the following lemma.

Lemma 4 ([25]). *The sum of subtree sizes over all light nodes is* $\mathcal{O}(n \log n)$.

We are now ready to complete the proof. Let $S_u \subseteq S$ denote the set of pairs (x, y), such that the lowest common ancestor of the leaves corresponding to suffixes $\mathsf{T}[x, n]\$$ and $\mathsf{T}[y, n]\$$ is u. Clearly, the paths from the root to the leaves corresponding to suffixes $\mathsf{T}[x, n]\$$ and $\mathsf{T}[y, n]\$$ pass from two distinct children of node u and then at least one of the two must be a light node. Therefore, $|S_u|$ is at most twice the sum of $\mathsf{size}(\cdot)$ over all light children of u. Since $|S| = \sum_u |S_u|$, we can bound $|S|$ by the sum of $\mathsf{size}(\cdot)$ over all light nodes in the suffix tree, which is $\mathcal{O}(n \log n)$ by Lemma 4. This completes the proof of Lemma 3.

4 Open Questions

We leave the following related questions unanswered:

1. Can we design an efficient $\mathcal{O}(n)$-word data structure for the rSUS problem?
2. Can we design an efficient solution for the k mismatches/edits variation of the rSUS problem, perhaps using the framework of [26]?
3. Can our reduction be extended to other types of string regularities?

References

1. Abedin, P., et al.: A linear-space data structure for Range-LCP queries in polylogarithmic time. In: Proceedings of Computing and Combinatorics - 24th International Conference, COCOON 2018, Qing Dao, China, 2–4 July 2018. pp. 615–625 (2018). https://doi.org/10.1007/978-3-319-94776-1_51
2. Allen, D.R., Thankachan, S.V., Xu, B.: A practical and efficient algorithm for the k-mismatch shortest unique substring finding problem. In: Shehu, A., Wu, C.H., Boucher, C., Li, J., Liu, H., Pop, M. (eds.) Proceedings of the 2018 ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics, BCB 2018, Washington, DC, USA, 29 August–01 September 2018. pp. 428–437. ACM (2018). https://doi.org/10.1145/3233547.3233564
3. Amir, A., Apostolico, A., Landau, G.M., Levy, A., Lewenstein, M., Porat, E.: Range LCP. J. Comput. Syst. Sci. **80**(7), 1245–1253 (2014). https://doi.org/10.1016/j.jcss.2014.02.010
4. Amir, A., Lewenstein, M., Thankachan, S.V.: Range LCP queries revisited. In: Iliopoulos, C., Puglisi, S., Yilmaz, E. (eds.) SPIRE 2015. LNCS, vol. 9309, pp. 350–361. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23826-5_33
5. Ayad, L.A.K., Pissis, S.P., Polychronopoulos, D.: CNEFinder: finding conserved non-coding elements in genomes. Bioinformatics **34**(17), i743–i747 (2018). https://doi.org/10.1093/bioinformatics/bty601
6. Bender, M.A., Farach-Colton, M.: The LCA problem revisited. In: Gonnet, G.H., Viola, A. (eds.) LATIN 2000. LNCS, vol. 1776, pp. 88–94. Springer, Heidelberg (2000). https://doi.org/10.1007/10719839_9
7. Berkman, O., Vishkin, U.: Recursive star-tree parallel data structure. SIAM J. Comput. **22**(2), 221–242 (1993). https://doi.org/10.1137/0222017
8. Chan, T.M., Nekrich, Y., Rahul, S., Tsakalidis, K.: Orthogonal point location and rectangle stabbing queries in 3-D. In: 45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, Prague, Czech Republic, 9–13 July 2018, pp. 31:1–31:14 (2018). https://doi.org/10.4230/LIPIcs.ICALP.2018.31
9. Farach, M.: Optimal suffix tree construction with large alphabets. In: 38th Annual Symposium on Foundations of Computer Science, FOCS 1997, Miami Beach, Florida, USA, 19–22 October 1997, pp. 137–143. IEEE Computer Society (1997). https://doi.org/10.1109/SFCS.1997.646102
10. Ganguly, A., Hon, W., Shah, R., Thankachan, S.V.: Space-time trade-offs for the shortest unique substring problem. In: 27th International Symposium on Algorithms and Computation, ISAAC 2016, Sydney, Australia, 12–14 December 2016, pp. 34:1–34:13 (2016). https://doi.org/10.4230/LIPIcs.ISAAC.2016.34
11. Ganguly, A., Hon, W., Shah, R., Thankachan, S.V.: Space-time trade-offs for finding shortest unique substrings and maximal unique matches. Theor. Comput. Sci. **700**, 75–88 (2017). https://doi.org/10.1016/j.tcs.2017.08.002
12. Ganguly, A., Patil, M., Shah, R., Thankachan, S.V.: A linear space data structure for range LCP queries. Fundam. Inform. **163**(3), 245–251 (2018). https://doi.org/10.3233/FI-2018-1741
13. Harel, D., Tarjan, R.E.: Fast algorithms for finding nearest common ancestors. SIAM J. Comput. **13**(2), 338–355 (1984). https://doi.org/10.1137/0213024
14. Haubold, B., Pierstorff, N., Möller, F., Wiehe, T.: Genome comparison without alignment using shortest unique substrings. BMC Bioinform. **6**, 123 (2005). https://doi.org/10.1186/1471-2105-6-123

15. Hon, W., Thankachan, S.V., Xu, B.: In-place algorithms for exact and approximate shortest unique substring problems. Theor. Comput. Sci. **690**, 12–25 (2017). https://doi.org/10.1016/j.tcs.2017.05.032
16. İleri, A.M., Külekci, M.O., Xu, B.: Shortest unique substring query revisited. In: Kulikov, A.S., Kuznetsov, S.O., Pevzner, P. (eds.) CPM 2014. LNCS, vol. 8486, pp. 172–181. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07566-2_18
17. Iliopoulos, C.S., Mohamed, M., Pissis, S.P., Vayani, F.: Maximal motif discovery in a sliding window. In: Gagie, T., Moffat, A., Navarro, G., Cuadros-Vargas, E. (eds.) SPIRE 2018. LNCS, vol. 11147, pp. 191–205. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00479-8_16
18. Inoue, H., Nakashima, Y., Mieno, T., Inenaga, S., Bannai, H., Takeda, M.: Algorithms and combinatorial properties on shortest unique palindromic substrings. J. Discrete Algorithms **52**, 122–132 (2018). https://doi.org/10.1016/j.jda.2018.11.009
19. Khmelev, D.V., Teahan, W.J.: A repetition based measure for verification of text collections and for text categorization. In: Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2003, pp. 104–110. ACM, New York (2003). https://doi.org/10.1145/860435.860456
20. Mieno, T., Inenaga, S., Bannai, H., Takeda, M.: Shortest unique substring queries on run-length encoded strings. In: Faliszewski, P., Muscholl, A., Niedermeier, R. (eds.) 41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016, Kraków, Poland, 22–26 August 2016. LIPIcs, vol. 58, pp. 69:1–69:11. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2016). https://doi.org/10.4230/LIPIcs.MFCS.2016.69
21. Mieno, T., Köppl, D., Nakashima, Y., Inenaga, S., Bannai, H., Takeda, M.: Compact data structures for shortest unique substring queries. CoRR abs/1905.12854 (2019), http://arxiv.org/abs/1905.12854
22. Pei, J., Wu, W.C.H., Yeh, M.Y.: On shortest unique substring queries. In: 2013 IEEE 29th International Conference on Data Engineering (ICDE), pp. 937–948. IEEE (2013)
23. Schleiermacher, C., Ohlebusch, E., Stoye, J., Choudhuri, J.V., Giegerich, R., Kurtz, S.: REPuter: the manifold applications of repeat analysis on a genomic scale. Nucleic Acids Res. **29**(22), 4633–4642 (2001). https://doi.org/10.1093/nar/29.22.4633
24. Schultz, D.W., Xu, B.: On k-mismatch shortest unique substring queries using GPU. In: Proceedings of Bioinformatics Research and Applications - 14th International Symposium, ISBRA 2018, Beijing, China, 8–11 June 2018, pp. 193–204 (2018). https://doi.org/10.1007/978-3-319-94968-0_18
25. Sleator, D.D., Tarjan, R.E.: A data structure for dynamic trees. In: Proceedings of the 13th Annual ACM Symposium on Theory of Computing, Milwaukee, Wisconsin, USA, 11–13 May 1981, pp. 114–122 (1981). https://doi.org/10.1145/800076.802464
26. Thankachan, S.V., Aluru, C., Chockalingam, S.P., Aluru, S.: Algorithmic framework for approximate matching under bounded edits with applications to sequence analysis. In: Proceedings of Research in Computational Molecular Biology - 22nd Annual International Conference, RECOMB 2018, Paris, France, 21–24 April 2018, pp. 211–224 (2018). https://doi.org/10.1007/978-3-319-89929-9_14

27. Tsuruta, K., Inenaga, S., Bannai, H., Takeda, M.: Shortest unique substrings queries in optimal time. In: Geffert, V., Preneel, B., Rovan, B., Štuller, J., Tjoa, A.M. (eds.) SOFSEM 2014. LNCS, vol. 8327, pp. 503–513. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-04298-5_44
28. Watanabe, K., Nakashima, Y., Inenaga, S., Bannai, H., Takeda, M.: Shortest unique palindromic substring queries on run-length encoded strings. In: Proceedings of Combinatorial Algorithms - 30th International Workshop, IWOCA 2019, Pisa, Italy, 23–25 July 2019, pp. 430–441 (2019). https://doi.org/10.1007/978-3-030-25005-8_35
29. Weiner, P.: Linear pattern matching algorithms. In: Proceedings of the 14th Annual Symposium on Switching and Automata Theory (SWAT 1973), pp. 1–11. IEEE Computer Society, Washington, DC (1973). https://doi.org/10.1109/SWAT.1973.13
30. Yao, A.C.: Space-time tradeoff for answering range queries (extended abstract). In: Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing, STOC 1982, pp. 128–136. ACM, New York (1982). https://doi.org/10.1145/800070.802185