Space-time tradeoff for the shortest unique substring problem

1 Problem Formulation

The shortest unique substring (SUS) is defined as the shortest substring U of text T such that there is only one occurrence of U in T.

2 Lemma 1

There is an algorithm that given integer parameters ℓ, r runs in $\mathcal{O}(\frac{n^2}{r-\ell})$ time and $\mathcal{O}(r-\ell)$ space and returns YES if the length of the shortest unique substring is less than ℓ, NO if the length of the shortest unique substring is greater than r, and an arbitrary answer otherwise.

Proof:
Let $\delta = r-\ell$. Consider the substrings $S_k = T[k\delta+1..k\delta+r]$ for $k = 0, 1, .., \left\lfloor \frac{|T|}{\delta} \right\rfloor$. We determine the uniqueness of each S_k in text T using a constant space, linear time pattern matching algorithm. If no S_k are found to be unique, then the algorithm returns NO. If one or more S_k are found to be unique, then the algorithm returns YES.

Case $|SUS| > r$:
If $|SUS| > r$, then by the definition of the shortest unique substring, no substring of T of length less than or equal to r will be unique in T. It follows that no S_k will be unique in T. In this case our algorithm returns NO, correctly.

Case $|SUS| < \ell$:
If $|SUS| < \ell$, then the shortest unique substring must be a substring of some S_k. This is because for all S_i such that S_{i+1} exists, the starting position of S_{i+1} minus the ending position of S_i equals $\ell - 1$. In other words, all S_k overlap by $\ell - 1$ characters, so if $|SUS| < \ell$, then the SUS must be a substring of some S_k. If the shortest unique substring is a substring of some S_k, then that S_k will be unique in T. Therefore, if $|SUS| < \ell$, then some S_k will be unique in T. In this case our algorithm returns YES, correctly. $\square$

3 Theorem 1

There is an algorithm that given a parameter τ, $1 \leq \tau \leq n$, runs in $\mathcal{O}(\frac{n^2}{\tau})$ time and $\mathcal{O}(1)$ space and returns a substring that is unique in T with length at most $|SUS| + \tau - 1$.

Proof:
We now use Lemma 1 to perform ternary search over the range R of possible values of $|SUS|$. We initialize R to be $[1, n]$. During each iteration of the ternary search we execute the algorithm in Lemma 1 setting ℓ and r approximately to 1/3 and 2/3 of R, to reduce the range by one third. We continue this until $|R| \leq \tau$. The final iteration of the ternary search will produce a unique substring of T with length at most $|SUS| + \tau - 1$. The time complexity is a geometric progression dominated by the final term $\mathcal{O}(\frac{n^2}{\tau})$. The space remains $\mathcal{O}(1)$. $\square$

4 Theorem 2

There is an algorithm that given a parameter τ, $1 \leq \tau \leq n$, computes SUS in $\mathcal{O}(\frac{n^2}{\tau})$ time and $\mathcal{O}(\tau)$ space.

Proof:
Theorem 1 yields a range $[a, b]$ such that $a \leq |SUS| \leq b$ and $b - a \leq \tau$.

Case $a = 1$:
Let $\delta = b - a$. Consider the substrings $S_k = T[k\delta+1..k\delta+b]$ for $k = 0, 1, .., \lfloor\frac{|T|}{\delta}\rfloor$. Note that there are $\mathcal{O}(n/\tau)$ substrings S_k, and $|S_k| = \mathcal{O}(\tau)$ for all k. The SUS must be a substring of some S_k. We can find the SUS by finding the shortest substring of some S_k that has exactly one occurrence in the entire list of substrings S_k.
For each substring S_k construct the suffix tree of S_k in $\mathcal{O}(\tau)$ space and $\mathcal{O}(\tau)$ time. Make $\mathcal{O}(\tau)$ counters for the $\mathcal{O}(\tau)$ leaves of S_k's suffix tree, using $\mathcal{O}(\tau)$ space. Set each counter to zero initially. Sequentially for every S_i such that $i \neq k$ construct the generalized suffix tree (gst) of S_k and S_i in $\mathcal{O}(\tau)$ time and $\mathcal{O}(\tau)$ space. For each leaf of S_k in the gst, set the corresponding counter to the string depth of the parent node + 1. If the string depth of the parent node + 1 is greater than the current value of the counter, set the counter to this new value. After constructing The gst of S_k and S_i, we construct the gst of S_{i+1} by removing S_i from the gst and replacing it with S_{i+1}. After comparing S_k to every S_i we take the smallest counter value as the shortest substring of S_k that has only one occurrence in the entirety of substrings S_i. By performing this search over all k we can find the shortest unique substring.
We are constructing $\mathcal{O}(\frac{n^2}{\tau^2})$ generalized suffix trees since we are comparing every S_k to every other substring of which there are $\mathcal{O}(\frac{n}{\tau})$. Each generalized suffix tree can be constructed in $\mathcal{O}(\tau)$ time. Therefore the time bound of this algorithm is

$\mathcal{O}(\frac{n^2}{\tau})$, and the space bound is $\mathcal{O}(\tau)$.

General Case:
If $a \leq 10\tau$ we can use the technique described above and adjust the multiplicative constants to preserve the time and space bounds.

If $a > 10\tau$, the solution is the exact same as this paper https://arxiv.org/pdf/1407.0522.pdf. $\square$

5 Theorem 3

Given a text T of length n from an alphabet Σ of size at least n^2, any deterministic RAM algorithm, which uses $\tau \leq \frac{n}{\log n}$ space to compute the shortest unique substring of some document must use time $\Omega(n\sqrt{\log(n/(\tau \log n))/\log\log(n/(\tau \log n))})$.

Proof:
The Element Distinctness Problem has the bound $\Omega(n\sqrt{\log(n/(\tau \log n))/\log\log(n/(\tau \log n))})$. We will now demonstrate that a similar problem, the Element Non-distinctness Problem has the same bound. The ENP takes n elements and decides if all the elements are non-distinct. If we let the elements considered in ENP be $T[i]$ for $i = 1, .., n$, then deciding ENP is equivalent to deciding if $SUS(T) = 1$.

(This proof is unfinished)