Minimal unique palindromic substrings after single-character substitution

Mitsuru Funakoshi, Takuya Mieno*

Kyushu University, Japan

*Current affiliation: Hokkaido University, Japan

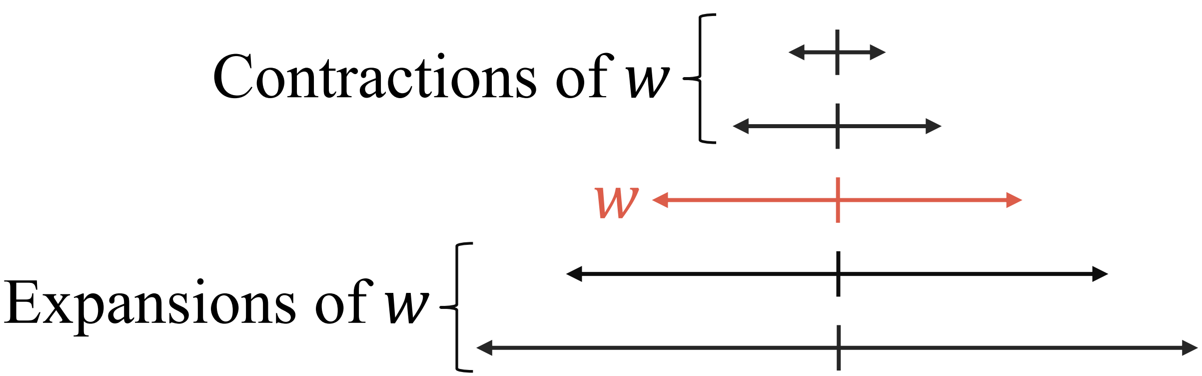
Palindromes

Palindrome is a string that reads the same forward and backward.

For a palindromic substring w of a string T,
a palindrome whose center is the same as that of w is called

- a **contraction** of w if it is shorter than w, and
- an **expansion** of w if it is longer than w.

T = a b b a a b b b a a b a b a b a b



Palindromes

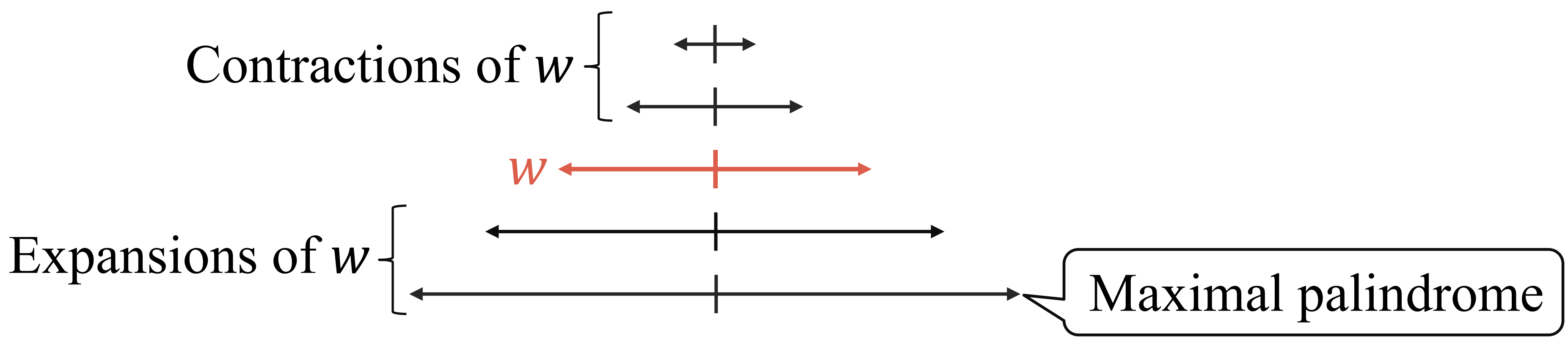
Palindrome is a string that reads the same forward and backward.

For a palindromic substring w of a string T,
a palindrome whose center is the same as that of w is called

- a **contraction** of w if it is shorter than w, and
- an **expansion** of w if it is longer than w.

If there are no expansions of u, u is called a maximal palindrome.

T = a b b a a b b b a a b a b a b a b



Minimal Unique Palindromic Substring

Definition [Inoue+, '18]

A palindromic substring $T[i..j]$ of a string T is a **minimal unique palindromic substring (MUPS)** of T if $T[i..j]$ is unique in T and $T[i+1..j-1]$ is repeating in T.

T = a b b a a b b b a a b a b a b a b

Minimal Unique Palindromic Substring

Definition [Inoue+, '18]
A palindromic substring $T[i..j]$ of a string T is a **minimal unique palindromic substring (MUPS)** of T if $T[i..j]$ is unique in T and $T[i+1..j-1]$ is repeating in T.

T = a b **b a a b** b **b a a b** a b a b a b

Minimal Unique Palindromic Substring

Definition [Inoue+, '18]
A palindromic substring $T[i..j]$ of a string T is a **minimal unique palindromic substring (MUPS)** of T if $T[i..j]$ is unique in T and $T[i+1..j-1]$ is repeating in T.

T = a b b a a b b b a a b a b a b a b

Theorem [Inoue+, '18]
For any string T of length n, $|\mathrm{MUPS}(T)| \leq n$.
$\mathrm{MUPS}(T)$ can be computed in $O(n)$ time.

$\mathrm{MUPS}(T)$: the set of MUPSs of a string T.

Our problem

> **Problem** (*MUPS_AFTER_EDIT*)
> Preprocessing input: string T
> Query input: single character substitution $\langle i, s \rangle$
> Query output: $\mathrm{MUPS}(T) \triangle \mathrm{MUPS}(T')$ (T': the edited string)

Substituting $T[i]$ with a character s.

Query input: $\langle i, s \rangle = \langle 7, \mathrm{a} \rangle$

i

T = a b b a a b $\boxed{\texttt{b}}$ b a a b a b a b

T' = a b b a a b $\boxed{\texttt{a}}$ b a a b a b a b

Query output: abba, bbb, bbaabb, bb, aababaa, abaaba

Removed Added

Our problem

Problem (*MUPS_AFTER_EDIT*)
Preprocessing input: string T
Query input: single character substitution $\langle i, s \rangle$
Query output: $\mathrm{MUPS}(T) \,\triangle\, \mathrm{MUPS}(T')$ (T': the edited string)

Substituting $T[i]$ with a character s.

Query input: $\langle i, s \rangle = \langle 9, \mathrm{b} \rangle$

i

T = a b b a a b b b $\boxed{\texttt{a}}$ a b a b a b

T' = a b b a a b b b $\boxed{\texttt{b}}$ a b a b a b

Query output: bbb, bbaabb, babab, aa, bbbb

Removed Added

Related work for “after one-edit” problems

	Static
Longest common substring	$O(n)$ time [Weiner, ’73]
Longest palindrome	$O(n)$ time [Manacher, ’75]
Longest Lyndon substring	$O(n)$ time [Duval, ’83]

Related work for "after one-edit" problems

	Static	One edit
Longest common substring	$O(n)$ time [Weiner, '73]	$\tilde{O}(n)$ space $\tilde{O}(1)$ time [Amir+, '17] [Abedin+, '18]
Longest palindrome	$O(n)$ time [Manacher, '75]	$O(n)$ space $\tilde{O}(1)$ time [Funakoshi+, '21]
Longest Lyndon substring	$O(n)$ time [Duval, '83]	$O(n)$ space $\tilde{O}(1)$ time [Urabe+, '18]

$\tilde{O}(f(n)) = O(f(n)\ \mathrm{polylog}(n))$

Related work for “after one-edit” problems

	Static	One edit	Dynamic
Longest common substring	$O(n)$ time [Weiner, ’73]	$\tilde{O}(n)$ space $\tilde{O}(1)$ time [Amir+, ’17] [Abedin+, ’18]	$O(n)$ space $\tilde{O}(1)$ time [Charalampopoulos+, ’20]
Longest palindrome	$O(n)$ time [Manacher, ’75]	$O(n)$ space $\tilde{O}(1)$ time [Funakoshi+, ’21]	$O(n)$ space $\tilde{O}(1)$ time [Amir & Boneh, ’19]
Longest Lyndon substring	$O(n)$ time [Duval, ’83]	$O(n)$ space $\tilde{O}(1)$ time [Urabe+, ’18]	$O(n)$ space $\tilde{O}(n^{1/2})$ time [Amir+, ’19]*

$\tilde{O}(f(n)) = O(f(n)\,\mathrm{polylog}(n))$

*Randomized algorithm

Related work for “after one-edit” problems

	Static	One edit	Dynamic
Longest common substring	$O(n)$ time [Weiner, ’73]	$\tilde{O}(n)$ space $\tilde{O}(1)$ time [Amir+, ’17] [Abedin+, ’18]	$O(n)$ space $\tilde{O}(1)$ time [Charalampopoulos+, ’20]
Longest palindrome	$O(n)$ time [Manacher, ’75]	$O(n)$ space $\tilde{O}(1)$ time [Funakoshi+, ’21]	$O(n)$ space $\tilde{O}(1)$ time [Amir & Boneh, ’19]
Longest Lyndon substring	$O(n)$ time [Duval, ’83]	$O(n)$ space $\tilde{O}(1)$ time [Urabe+, ’18]	$O(n)$ space $\tilde{O}(n^{1/2})$ time [Amir+, ’19]*
MUPS	$O(n)$ time [Inoue+, ’18]	$O(n)$ space $\tilde{O}(1)$ time **[This work]**	-

$\tilde{O}(f(n)) = O(f(n)\ \mathrm{polylog}(n))$

*Randomized algorithm

Main results

Theorem 1

The number $\boldsymbol{d}$ of changes of MUPSs is $\boldsymbol{O(\log n)}$ after single character substitution.

Theorem 2

MUPS_AFTER_EDIT can be solved in the following query time after $O(n)$ time and space preprocessing:

	Alphabet size σ	Query time
Algorithm 1 (for large σ)	$O(\mathrm{poly}(n))$	$O(\log \sigma + (\log \log n)^2 + d)$
	$O(n)$	$O((\log \log n)^2 + d)$
Algorithm 2 (for small σ)	$O(\log n)$	$O(\log \log n + d)$
	$O(1)$	$O(1 + d)$

Main results

Theorem 1

The number $\boldsymbol{d}$ of changes of MUPSs is $\boldsymbol{O(\log n)}$ after single character substitution.

Theorem 2

MUPS_AFTER_EDIT can be solved in the following query time after $O(n)$ time and space preprocessing:

	Alphabet size σ	Query time
Algorithm 1 (for large σ)	$O(\text{poly}(n))$	$O(\log\sigma + (\log\log n)^2 + d)$
	$O(n)$	$O((\log\log n)^2 + d)$
Algorithm 2 (for small σ)	$O(\log n)$	$O(\log\log n + d)$
	$O(1)$	$O(1 + d)$

Main results

Theorem 1

The number $\boldsymbol{d}$ of changes of MUPSs is $\boldsymbol{O(\log n)}$ after single character substitution.

Theorem 2

MUPS_AFTER_EDIT can be solved in the following query time after $O(n)$ time and space preprocessing:

	Alphabet size σ	Query time
Algorithm 1 (for large σ)	$O(\mathrm{poly}(n))$	$O(\log \sigma + (\log\log n)^2 + d)$
	$O(n)$	$O((\log\log n)^2 + d)$
Algorithm 2 (for small σ)	$O(\log n)$	$O(\log\log n + d)$
	$O(1)$	$O(1 + d)$

Main results

Theorem 1

The number $\boldsymbol{d}$ of changes of MUPSs is $\boldsymbol{O(\log n)}$ after single character substitution.

Theorem 2

MUPS_AFTER_EDIT can be solved in the following query time after $O(n)$ time and space preprocessing:

	Alphabet size σ	Query time
Algorithm 1 (for large σ)	$O(\mathrm{poly}(n))$	$O(\log\sigma + (\log\log n)^2 + d)$
	$O(n)$	$O((\log\log n)^2 + d)$
Algorithm 2 (for small σ)	$O(\log n)$	$O(\log\log n + d)$
	$O(1)$	$O(1 + d)$

of changes of MUPSs

> **Theorem 1**
> The number $\boldsymbol{d}$ of changes of MUPSs is $\boldsymbol{O(\log n)}$.

We show # of MUPSs to be **removed** is $O(\log n)$.

We categorize MUPSs to be removed into three types:

- Type 1: covers the editing position i.
- Type 2: does not cover i and is repeating in T'.
- Type 3: does not cover i and is unique but not minimal in T'.

$$T \;=\; \mathtt{a\ b\ b\ a\ a\ b}\ \overset{i}{\boxed{\mathtt{b}}}\ \mathtt{b\ a\ a\ b\ a\ b\ a\ b}$$

Type 3 Type 2

Type 1

$$T' \;=\; \mathtt{a\ b\ b\ a\ a\ b}\ \boxed{\mathtt{a}}\ \mathtt{b\ a\ a\ b\ a\ b\ a\ b}$$

of changes of MUPSs

Theorem 1
The number $\boldsymbol{d}$ of changes of MUPSs is $\boldsymbol{O(\log n)}$.

We show # of MUPSs to be **removed** is $O(\log n)$.
We categorize MUPSs to be removed into three types:

- Type 1: covers the editing position i.
- Type 2: does not cover i and is repeating in T'.
- Type 3: does not cover i and is unique but not minimal in T'.

i

T = a b b a a b $\boxed{\text{b}}$ b a a b a b a b

Type 3 Type 2

Type 1

T' = a b b a a b $\boxed{\text{a}}$ b a a b a b a b

of changes of MUPSs

Theorem 1

The number $\boldsymbol{d}$ of changes of MUPSs is $\boldsymbol{O(\log n)}$.

We show # of MUPSs to be **removed** is $O(\log n)$.

We categorize MUPSs to be removed into three types:

- Type 1: covers the editing position i.
- Type 2: does not cover i and is repeating in T'.
- Type 3: does not cover i and is unique but not minimal in T'.

i

T = a b b a a b $\boxed{\text{b}}$ b a a b a b a b

Type 3

Type 2

Type 1

T' = a b b a a b $\boxed{\text{a}}$ b a a b a b a b

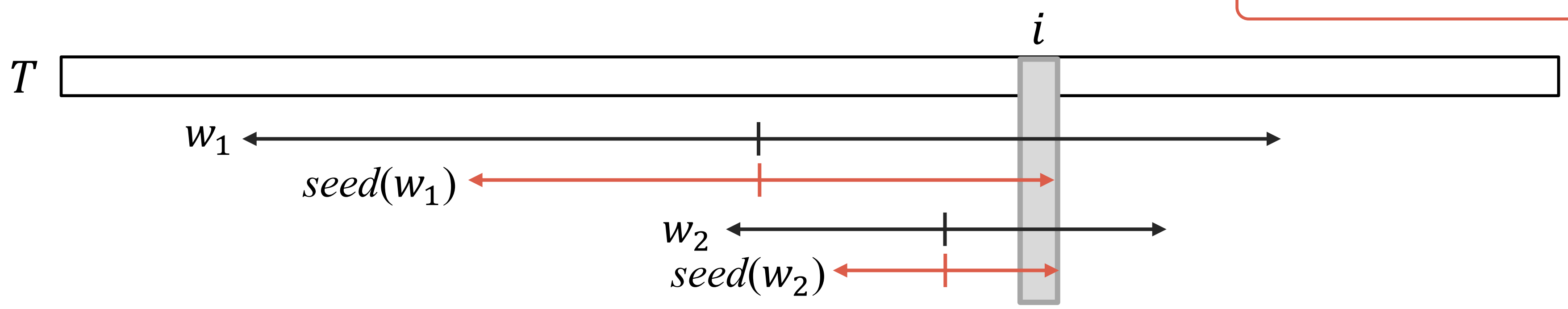
of MUPSs of Type 1

Type 1: covers i.

I focus on the MUPSs of Type 1 centered before i.

Such MUPS w is an expansion of some palindromic suffix of $T[1..i]$.

seed of MUPS w



Seeds of MUPSs are different from each other.

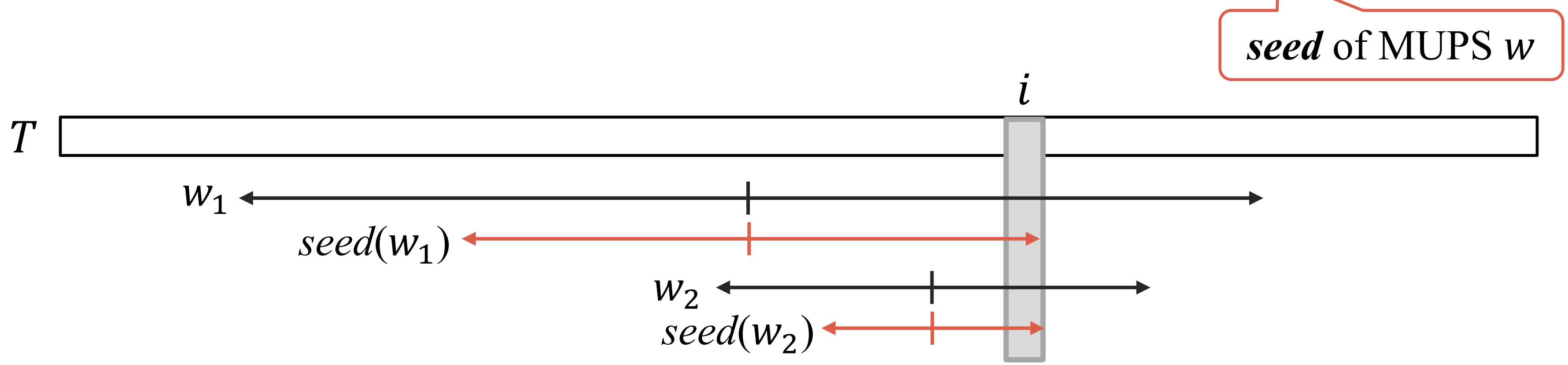
Thus, # of seeds is equal to # of MUPSs of Type 1.

of MUPSs of Type 1

Type 1: covers i.

I focus on the MUPSs of Type 1 centered before i.

Such MUPS w is an expansion of some palindromic suffix of $T[1..i]$.



Lemma 1 [Apostolico+, ’95]

The set of palindromic suffixes of $T[1..i]$ is divided into $O(\log i)$ groups w.r.t. their smallest period.

Claim 1

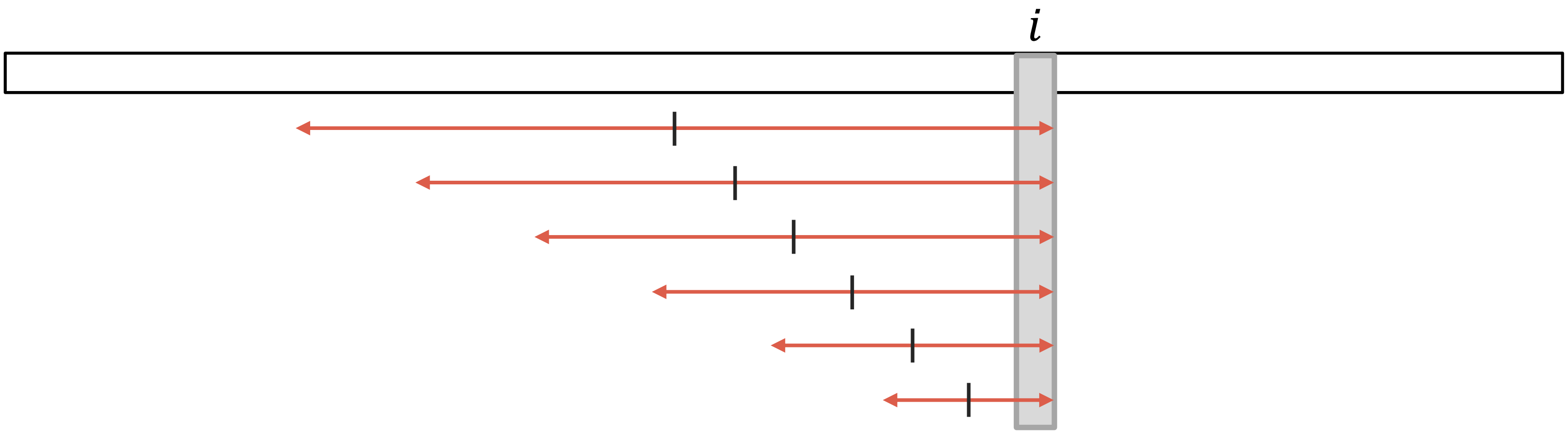
The number of seeds in each group is at most two.

If Claim 1 holds, then # of MUPSs of **Type 1** is $\boldsymbol{O(\log n)}$.

Proof of Claim 1

Claim 1

The number of seeds in each group is at most two.

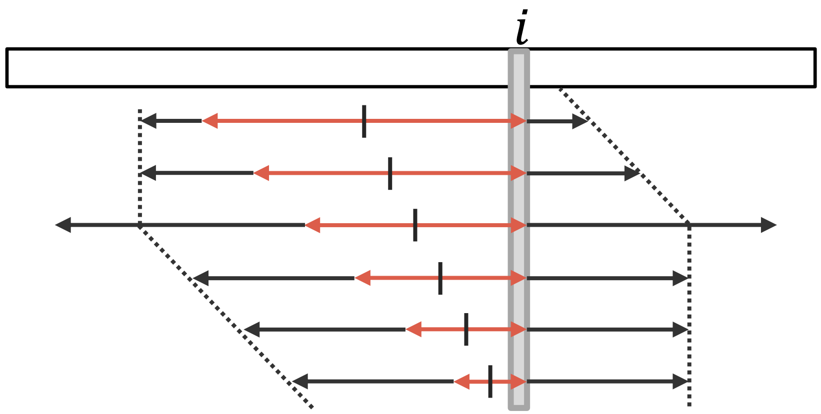


Now we consider the maximal expansions of palindromic suffixes, namely, maximal palindromes.

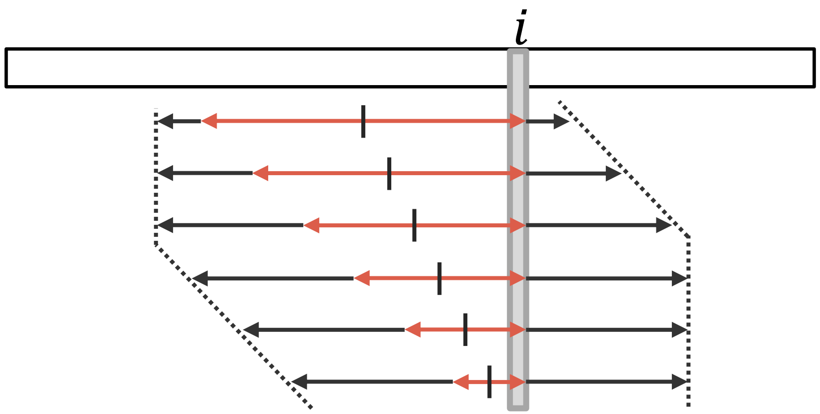
Proof of Claim 1

Claim 1

The number of seeds in each group is at most two.



or

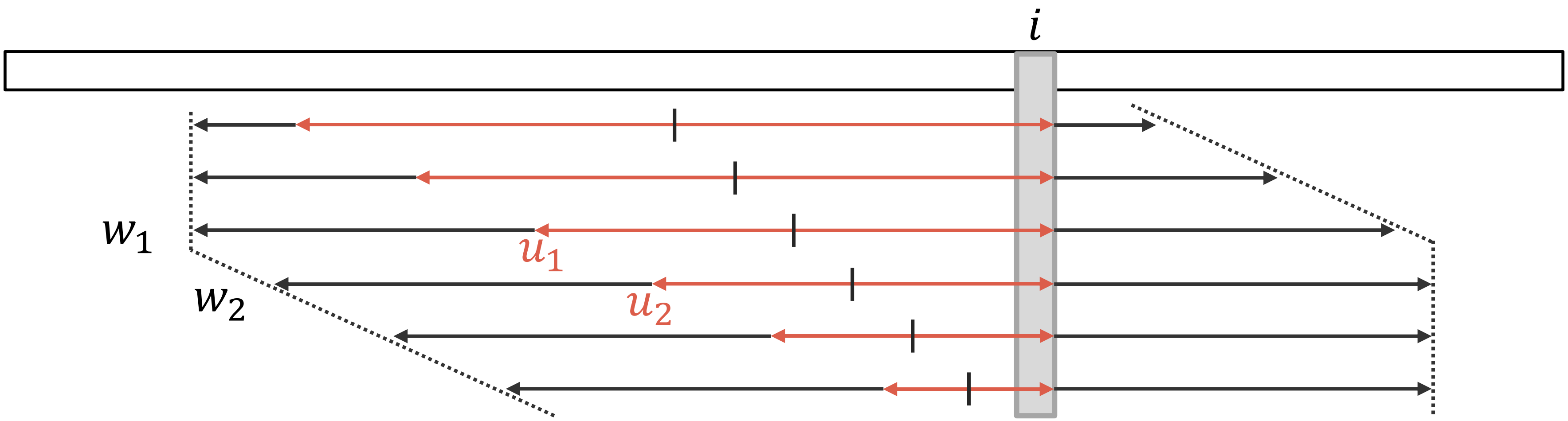


Now we consider the maximal expansions of palindromic suffixes, namely, maximal palindromes.

Due to periodicity, the expansions shape like this figure.

Proof of Claim 1

> Claim 1
> The number of seeds in each group is at most two.



- Any palindrome which is an expansion of palindromic suffixes is contained in w_1 or w_2.
- Any expansion of palindromic suffixes excluding u_1 and u_2 is repeating.

Thus, only u_1 and u_2 can be seeds.

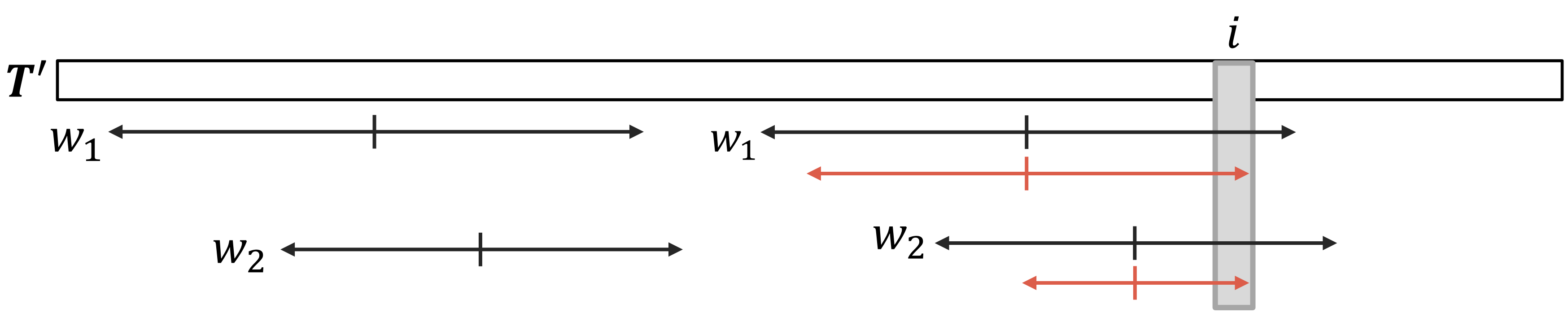
of MUPSs of Type 2

Type 2: does not cover i and is repeating in T'.

MUPS w of Type 2 has a new occurrence covering i in $\boldsymbol{T'}$.

Such an occ. is an expansion of a palindromic suffix of $T'[1..i]$.

Similar to the proof of Type-1, we reduce the problem of counting # of MUPSs of Type 2 to that of counting # of the palindromic suffixes corresponding to them.



We can prove that # of MUPSs of **Type 2** is $\boldsymbol{O(\log n)}$.

of changes of MUPSs

Similarly, we can also prove that # of MUPSs of **Type 3** is $\boldsymbol{O(\log n)}$.

From the above, # of MUPSs to be removed is $O(\log n)$.
By symmetry, # of MUPSs to be added is also $O(\log n)$.
Then we obtain Theorem 1.

Theorem 1
The number $\boldsymbol{d}$ of changes of MUPSs is $\boldsymbol{O(\log n)}$.

We have shown this upper bound is tight after submission.

Main results

Theorem 1

The number d of changes of MUPSs is $O(\log n)$ after single character substitution.

Theorem 2

MUPS_AFTER_EDIT can be solved in the following query time after $O(n)$ time and space preprocessing:

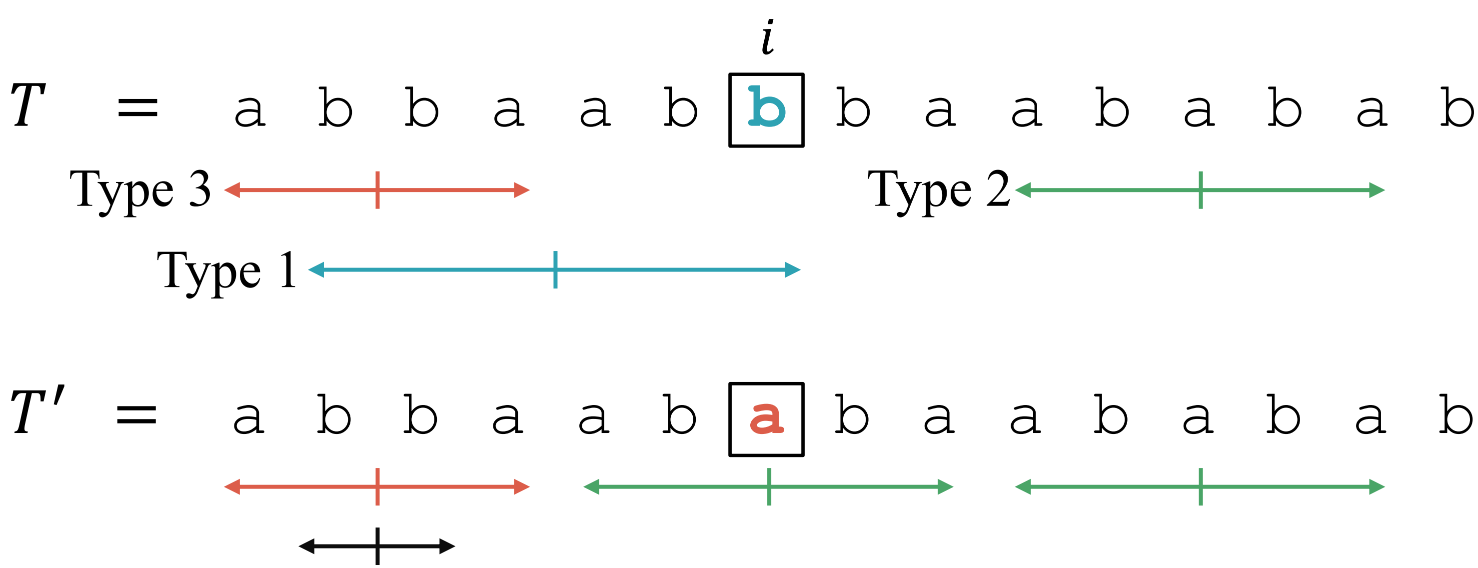
Alphabet size σ	Query time	
$O(\mathrm{poly}(n))$	$O(\log\sigma + (\log\log n)^2 + d)$	Using path-tree LCE query on EERTREE
$O(n)$	$O((\log\log n)^2 + d)$	
$O(\log n)$	$O(\log\log n + d)$	Using NCA query on Suffix Tree
$O(1)$	$O(1 + d)$	

Algorithm for $\sigma \in O(1)$

We separately compute MUPSs to be removed or added.
I will show how to compute MUPSs to be **removed**.

We use same categorizations of MUPSs to be removed:

- Type 1: covers i.
- Type 2: does not cover i and is repeating in T'.
- Type 3: does not cover i and is unique but not minimal in T'.

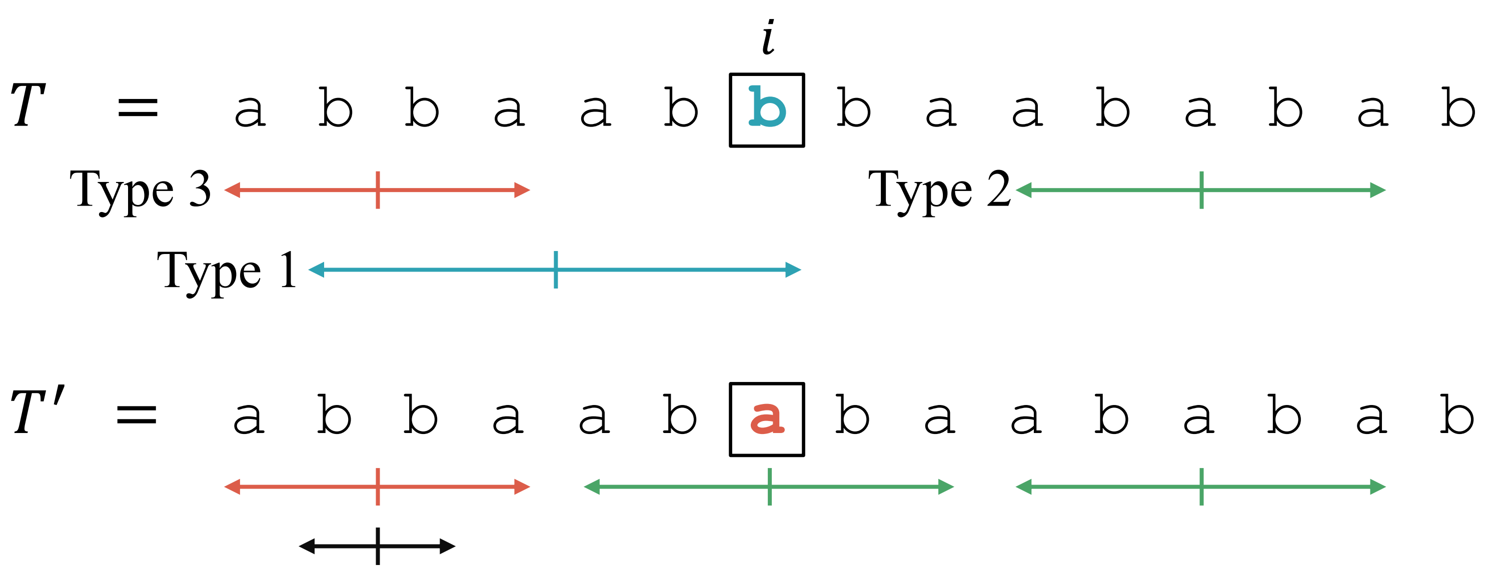


Algorithm for $\sigma \in O(1)$

We separately compute MUPSs to be removed or added.
I will show how to compute MUPSs to be **removed**.

We use same categorizations of MUPSs to be removed:

- Type 1: covers i.
- Type 2: does not cover i and is repeating in T'.
- Type 3: does not cover i and is unique but not minimal in T'.



Computing MUPSs of Type 2

<u>Type 2: does not cover i and is repeating in T'.</u>

MUPS w of Type 2 satisfies the following properties in $\boldsymbol{T'}$:

- \# of occurrences of w that covers i is at least 1.
- \# of occurrences of w that does not cover i is 1.

Further, we categorize MUPSs of Type 2 into two sub-types:

2-1: at least one occurrence of w covers i by its arm.
2-2: the only occurrence of w covering i is centered at i.

i

T = a b a a b a a b c a b a a $\boxed{\texttt{c}}$ a a b c

Type 2-1

Type 2-2

T' = a b a a b a a b c a b a a $\boxed{\texttt{b}}$ a a b c

Observation for MUPSs of Type 2-1

Type 2-1: at least one occurrence of w covers i by its arm.

Let j be the starting position of an occurrence of w such that its right-arm covers i in $\boldsymbol{T'}$.

$$\begin{array}{cccccccccccccccccccc} & & & & & & & & & & j & & & & i & & & & \\ \boldsymbol{T'} & = & \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{a} & \mathtt{b} & \mathtt{c} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{a} & \boxed{\mathtt{b}} & \mathtt{a} & \mathtt{a} & \mathtt{b} & \mathtt{c} \end{array}$$

w $\qquad$ w

Observation for MUPSs of Type 2-1

Type 2-1: at least one occurrence of w covers i by its arm.

Let j be the starting position of an occurrence of w such that its right-arm covers i in $\boldsymbol{T'}$.

The substring of length $|w|$ starting at j is a 1-mismatch palindrome whose left-arm matches that of w in $\boldsymbol{T}$.

$$
\begin{array}{cccccccccccccccccc}
 & & & & & & & & & & j & & & i & & & & \\
T & = & \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{a} & \mathtt{b} & \mathtt{c} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{a} & \boxed{\mathtt{c}} & \mathtt{a} & \mathtt{a} & \mathtt{b} & \mathtt{c} \\
 & w & & & & & & & & & & & & & & & & \\
T' & = & \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{a} & \mathtt{b} & \mathtt{c} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{a} & \boxed{\mathtt{b}} & \mathtt{a} & \mathtt{a} & \mathtt{b} & \mathtt{c} \\
 & w & & & & & & & & w & & & & & & & &
\end{array}
$$

Preprocessing for MUPSs of Type 2-1

Enumerate all 1-mismatch palindromes whose left- or right-arm matches that of some MUPS.

For each of the 1-mismatch palindromes, store the corresponding MUPS w with the key $(q, T[p])$, where q is the mismatched position and p is the position on w corresponding to q.

$$T = \texttt{a b a a}\ \overset{p}{\texttt{b}}\ \texttt{a a b c}\ \overset{j}{\texttt{a}}\ \texttt{b a a}\ \overset{q}{\boxed{\texttt{c}}}\ \texttt{a a b c}$$

w

The total time complexity is proportional to the total sum of occurrences of arms of all MUPSs.

Lemma 2
The total sum of occurrences of arms of all MUPSs is $O(n)$.

Thus, the above operations can be processed in $O(n)$ time and space.

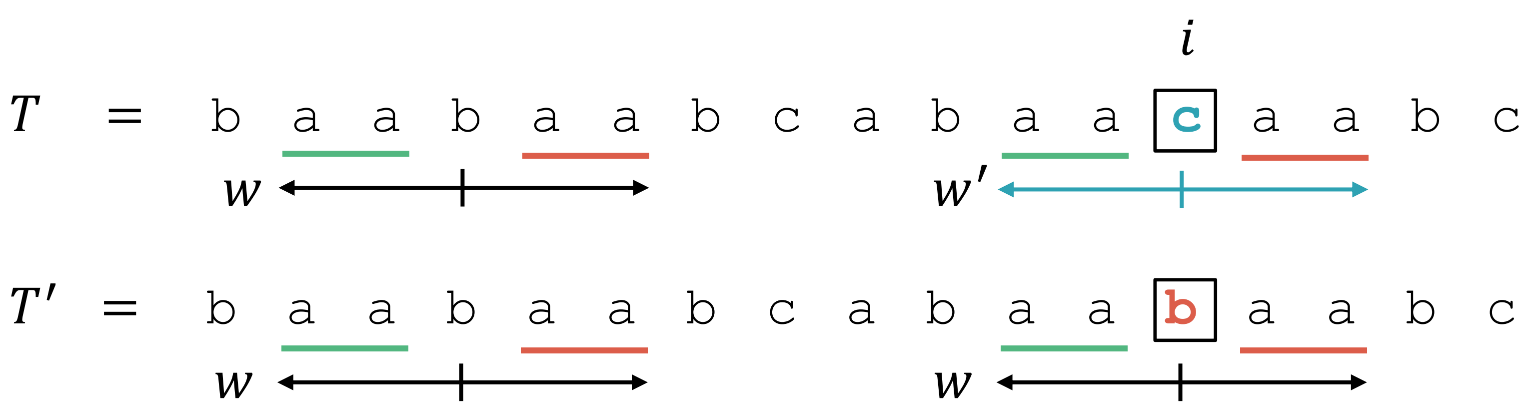
Observation for MUPS of Type 2-2

<u>Type 2-2: the only occurrence of w covering i is centered at i.</u>

There exists at most one MUPS of Type 2-2 for a query $\langle i, s \rangle$.
Also, such a MUPS w is an odd-palindrome.

If there exists such MUPS w for a query $\langle i, s \rangle$,
then there is a palindrome w' centered i such that $|w| = |w'|$ in T.
They differ only in the center character.
$\rightarrow$ The right-arm of w occurs at position $i + 1$ in both T and T'.

We design our algorithm by focusing on
occurrences of the right-arm of each MUPS in T.



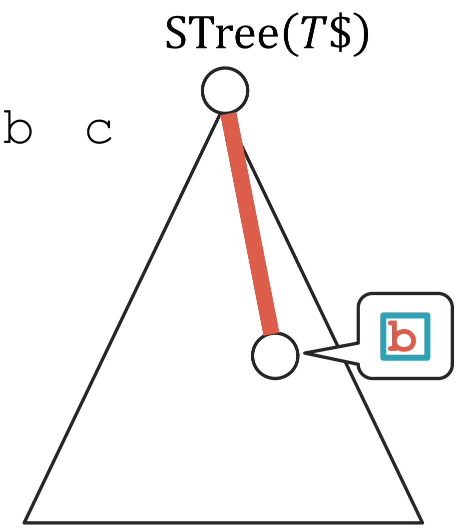
Preprocessing for MUPS of Type 2-2

We first construct the suffix tree STree(T\$).

For each odd MUPS in T, we make the locus of the right-arm explicit on STree(T\$) and label the node with the center character.

T = b a a b a a b c a b a a c a a b c

w



Preprocessing for MUPS of Type 2-2

We first construct the suffix tree STree(T\$).

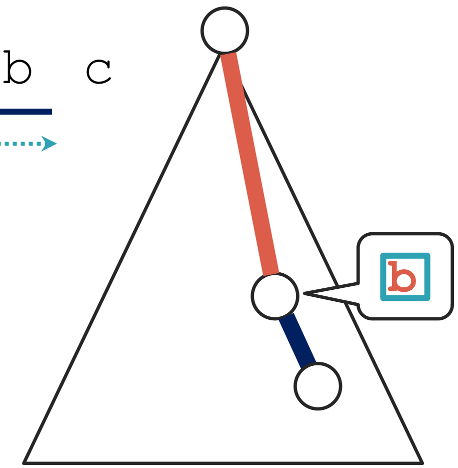
For each odd MUPS in T, we make the locus of the right-arm explicit on STree(T\$) and label the node with the center character.
Also, for each odd maximal palindrome,
we make the locus of the right-arm explicit on STree(T\$).

T = b a a $\boxed{\texttt{b}}$ a a b c a b a a c a a b c

w

odd maximal palindrome

STree(T\$)

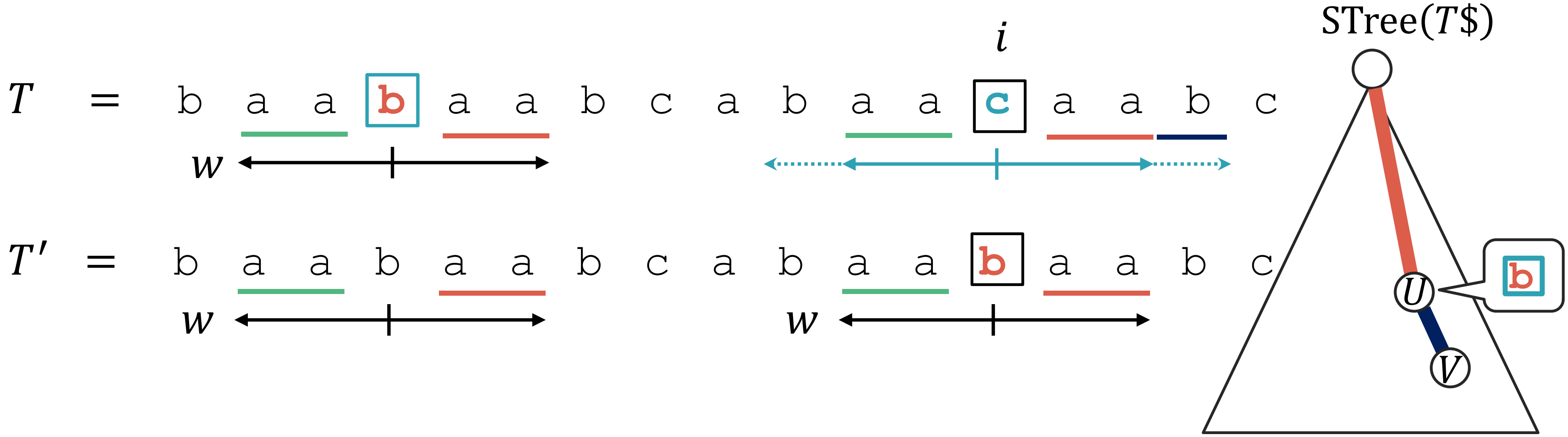


The above operations can be processed in $O(n)$ time and space by using weighted ancestor queries, Manacher's algorithms, and so on.

Query for MUPS of Type 2-2

Given a substitution $\langle i, s \rangle$,
we compute the nearest ancestor U labeled by s of the node V corresponding to the right-arm of the maximal palindrome centered at i.

If such U exists, $\mathrm{str}(U)^R \cdot s \cdot \mathrm{str}(U)$ is a MUPS of Type 2-2.

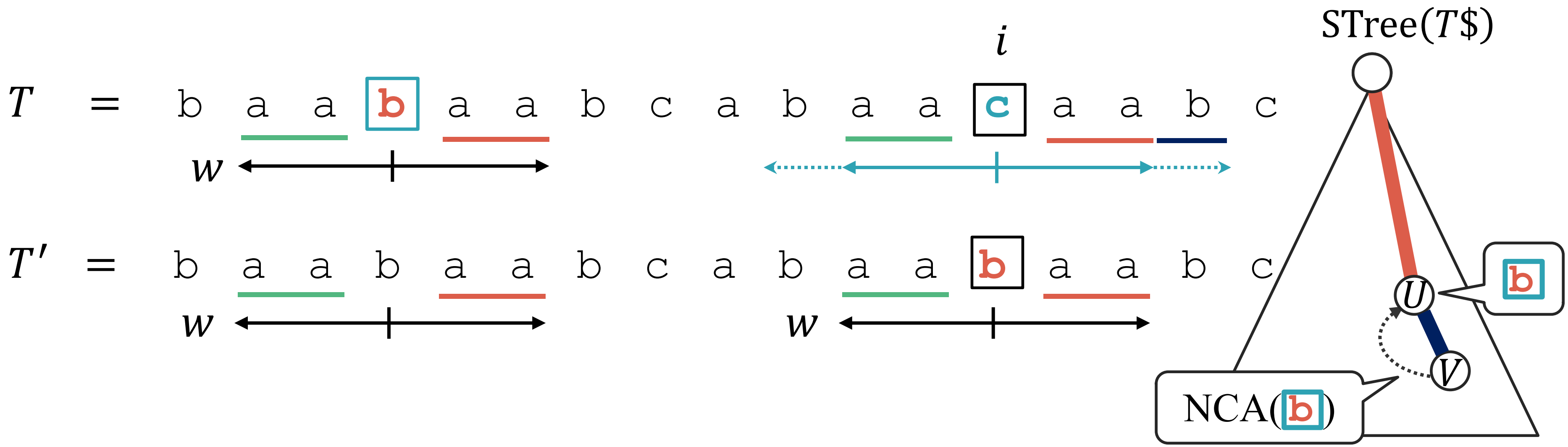


Query for MUPS of Type 2-2

Given a substitution $\langle i, s \rangle$,
we compute the nearest ancestor U labeled by s of the node V corresponding to the right-arm of the maximal palindrome centered at i.

Nearest Colored Ancestor (NCA)

If such U exists, $\mathrm{str}(U)^R \cdot s \cdot \mathrm{str}(U)$ is a MUPS of Type 2-2.



If # of colors is $O(\log n)$, any NCA query can be answered in $O(1)$ time after $O(n)$-time preprocessing [Bille+, ’15][Charalampopoulos+, ’21].
Thus, the MUPS of Type 2-2 can be computed in $O(1)$ query time.

Summary and future work

Our results

- The number d of changes of MUPSs is $O(\log n)$.
- *MUPS_AFTER_EDIT* can be solved in the following query time after $O(n)$ time and space preprocessing:

	Alphabet size σ	Query time
Algorithm 1 (for large σ)	$O(\mathrm{poly}(n))$	$O(\log\sigma + (\log\log n)^2 + d)$
	$O(n)$	$O((\log\log n)^2 + d)$
Algorithm 2 (for small σ)	$O(\log n)$	$O(\log\log n + d)$
	$O(1)$	$O(1 + d)$

Future work

- Insertions and deletions?
- Fully dynamic algorithm?
  - Can the techniques in [Amir and Boneh, '19] be utilized?