Finding All the Set-Maximal Matches over Non-Interactive Secure Computation

Yuki Koike[1,2,a)] Takaaki Nishimoto[1,b)] Yasuo Tabei[1] Jun Sakuma[1,2]

Abstract: In this paper, we propose an efficient non-interactive privacy-preserving algorithm for the set-maximal match problem. The set-maximal match problem is, given a pattern string and a set of strings, to enumerate all of the maximal segments in which the pattern and some string in the set have the same substring. The motivation of this problem arises from genetics. We show our method runs asymptotically faster than the existing method in both of theoretical time complexity and experiments.

Keywords: set-maximal match, non-interactive secure computation, TFHE, fully homomorphic encryption

1. Introduction

String problems are often of interest in genetic research because many important problems in this field can be formulated as a string problem, considering genetic sequences as strings. The SMM problem is one of such problems drawing attention. The goal of the problem is, given a string called pattern and a set of strings called dictionary, to find the set of longest substrings which occur in both of the pattern and some string in the dictionary. These substrings are called *set-maximal matches* (SMMs).

In genetics, a DNA segment shared by two or more people is called identical-by-decent (IBD) if it is inherited from a common ancestor without recombination. The detection of IBD segments has a broad range of applications, including genotype imputation [17, 21], haplotype phasing [17], heritability estimation [26, 6], and inferring demographic history [7, 16].

The SMM problem arises from these motivations for finding IBD segments; that is, SMMs among aligned genetic sequences are candidates for IBD segments.

The naive algorithm of enumerating all of the SMMs has a time complexity quadratic in the length of strings. This tends to be the crucial bottleneck of performance because the length of genetic sequences of interest often becomes more than tens of thousands. Up to the present, some algorithms running faster than the naive one have been proposed for the SMM problem [18, 19, 11]. In particular, Durbin proposed an optimal algorithm, offering the data structure called *positional BWT* (PBWT) [11].

When we apply the SMM problem to genetic studies, naturally possible is the situation where two different parties have a pattern and a dictionary, respectively, and the party who possesses the pattern would like to inquire of the other party all of the SMMs. In other words, we consider the particular case where a client queries an external genome database with his genetic sequence to obtain SMMs. These circumstances would occur, for example, in studies involving multiple institutions or in the direct-to-consumer services of DNA testing. For convenience, we call this problem setting as the two-party SMM problem.

Nowadays it is widely known that personal genome is so sufficient personal information as to identify individuals [20, 13], and what is worse, it has the exclusive characteristic of conceivably disclosing more confidential information such as the disease risk and blood relationship of individuals or even their progenitors and progeny [14, 3]. These concerns indicate that it is desirable to provide the way of solving the two-party SMM problem in a privacy-preserving manner.

Since this peril also threatens other topics concerning genome, several groups have engaged in the similar works of protecting genomic data privacy so far. Blanton and Aliasgari developed a protocol for securely searching a DNA sequence against DNA profiles represented by finite automata [4]. Similarly, Sasakawa et al. proposed oblivious protocols for NFA evaluation and showed as an example the privacy-preserving detection of viral infection with regular expression [29]. As to GWAS, there have already been many works devoted on securely performing statistics, including Kamm et al. [15], Lu et al. [23], Tkachenko et al. [32].

We call the following problem setting *the privacy-preserving SMM problem* (the pp-SMM problem). There are a server holding a dictionary and a client holding a pattern. As the same as the regular two-party SMM problem, the two parties communicate messages until the client

[1] RIKEN Center for Advanced Intelligence Project
[2] University of Tsukuba
[a)] yuki.koike@mdl.cs.tsukuba.ac.jp
[b)] takaaki.nishimoto@riken.jp

Table 1 Complexities of algorihtms for the SMM problem (provided that the size of alphabet is constant)

	Durbin [11]	Ours in plaintext	Shimizu et al. [30]	Ours in ciphertext
time complexity	amortized $O(n)$	$O(n)$	$O(n^2m)$	$O(nm(1+\frac{\log n}{\log m}))$
space complexity	$O(nm)$	$O(nm)$	$O(nm)$	$O(nm(1+\frac{\log n}{\log m}))$
round complexity	—	—	$O(n^2)$	$O(1)$

gets all of the SMMs between the dictionary and pattern. In addition, we require that during the whole process,all of the information, including the resultant SMMs must be kept secret to the server. Similarly, the client should obtain nothing but the information which can be deduced from the SMMs.

In this problem setting, we assume that both parties are "honest-but-curious", which means they are cooperative and genuinely perform the assigned operations, but might still try to retrieve any information they should not be able to acquire.

Table 1 shows the comparison of complexities among the algorithms related to our work, where n is the length of strings and m is the number of strings in the dictionary. We notice that the algorithms for the SMM problem and pp-SMM problem are put together in the table for the sake of explanation. The row of round complexity shows the asymptotic number of communication two parties exchange during the algorithm. Since this complexity is concerned only in the scope of secure computation, we omit the round complexities of plain-text algorithms.

To the best of our knowledge, despite the fact that there exists an optimal algorithm for the SMM problem, there is no known algorithm for the pp-SMM problem, which runs as fast as the optimal one of the SMM problem [11]. The only existing method for the pp-SMM problem is proposed by Shimizu et al. [30]. They represented PBWT and some parts of Durbin's algorithm in secure computation, applying oblivious transfer [27] recursively. Its time complexity is, however, quadratic in n, while Durbin's one is linear. This difference comes from the difficulty of fitting plain-text algorithms to secure computation keeping their time complexity the same. To be more precise, Durbin's algorithm almost certainly cannot be expressed efficiently in secure computation due to its dependence on amortization. Generally, amortized algorithms achieve their efficiency taking advantage of the fact that not every operation takes the worst-case cost. On the other hand, such cost saving is impossible in secure computation. Since we cannot know the concrete values of input in secure computation, there is no way to find out which operation takes the worst-case running time and we have no choice but to assume all the operations are the worst case. This would be the exact reason why the algorithm by Shimizu et al. suffered the degradation of time complexity compared to Durbin's one. This quadratic factor is non-negligible when the length of strings becomes large, and thus their algorithm is far from being optimal as for n. Because genetic studies often require, and call for the ability of, analyzing large DNA sequences [19, 31], it is unpreferable to have such heavy factor.

Another issue in the existing algorithm lies in its multi-round protocol (i.e., the algorithm offers interactive secure computation). Shimizu et al. solved the problem combining PBWT with oblivious transfer, which inevitably causes the communication between the two parties many times. The number of the communications scales quadratically with the length of the strings. This could result in inefficiency when the client and server need to spend a high cost for communication. Furthermore multi-round algorithms require the synchronization between the two parties; clients must participate in the computation and hold their system resources available during it, which could make it difficult for them to make effective use of such computing services.

The goal of this work is to eliminate these two issues. With regard to time complexity, we first modify and "deamortize" Durbin's algorithm. Introducing another data structure called *compact trie*, we succeed in "deamortization", namely, proposing the plain-text algorithm which does not depend on amortized complexity analysis, different from Durbin's one. Moreover, through this "deamortization", we find the existence of automaton which enables enumerating all of the SMMs. This achievement can be formally stated with the following theorems.

▶ **Theorem 1.** *Given a dictionary D of m strings of length n over an alphabet of size σ, there exists a deterministic finite automaton of size $O(nm\sigma)$, with which one can enumerate all of the SMMs between D and any given pattern where σ is the alphabet size. The time complexity of evaluating the automaton is $O(n)$.*

▶ **Theorem 2.** *The automaton described in Theorem 1 can be constructed in $O(nm\sigma)$ time for a dictionary D of m strings of length n given.*

In order to resolve the second issue of round complexity, we present the first non-interactive algorithm, which solves the pp-SMM problem in a single round. Our idea is to employ *fully homomorphic encryption* (FHE) (e.g., [12]). FHE is a form of encryption supporting addition and multiplication on ciphertexts. In recent years, FHE is gaining significant attention as a promising candidate for solutions to privacy preservation issues. For example, FHE is very applicable in the sphere of outsourcing statistical calculation or machine learning [33, 25, 22].

Some of FHE schemes particularly allow any party to evaluate an arbitrary Boolean circuit for a given input without the party learning the input and output, that is,

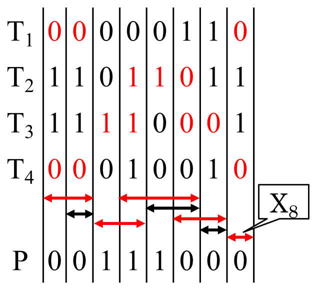

Figure 1 An example for SMMs and SRMMs. Each arrow starting at the i-th character of P represents the SRMM starting at position i. The red arrows are the SMMs between D and P. Red substrings starting at position i on texts represents the matched string by the SMM starting at the same position. X_8 represents the SMM starting at position 8.

with the input and output encrypted. This implies that Boolean circuits outputting the SMMs for a given pattern can solve the pp-SMM problem in a single round. Combining this implication with the above theorems, we finally produce such a circuit from the automaton in order to realize the efficient solution of the pp-SMM problem. Later we show our algorithm runs asymptotically faster than the algorithm by Simizu et al. in both of theoretical time complexity and experiments.

The following theorem describes our final result.

▶ **Theorem 3.** *Given a dictionary D of m strings of length n, there exists a Boolean circuit of size $O(\frac{nm\log\sigma\log(nm\log\sigma)}{\log m})$ which outputs all of the SMMs for any given pattern.*

Considering the fact that circuits without memory are naturally data-oblivious (i.e., do not change their behavior depending on input), Theorem 3 implies that the pp-SMM problem can be solved in $O(\frac{nm\log\sigma\log(nm\log\sigma)}{\log m})$ by evaluating such a circuit over FHE schemes.

2. Preliminaries

2.1 Strings

Let Σ be an ordered alphabet of size σ. For a string T of length n over Σ, let $|T|$ be the length of T, $T[i]$ be the i-th character of T, and $T[i..j]$ be the substring of T that begins at position i and ends at position j. The $T[i..]$ denotes the suffix of T beginning at position i, i.e., $T[i..n]$. Let $[i,j]$ be $\{i,i+1,\dots,j\}$ for two integers i and j ($i \le j$). For two strings T_1 and T_2, let $T_1 \cdot T_2$ be the concatenated string of T_1 and T_2, i.e., $T_1 \cdot T_2 = T_1[1], T_1[2], \dots, T_1[|T_1|], T_2[1], T_2[2], \dots, T_2[|T_2|]$.

2.2 The SMM problem

Let $D = T_1, \dots, T_m$ be m strings of each length n over Σ called *dictionary*, and P be a string of length n over Σ called *pattern*. We say that an interval $[a..b]$ on $[1..n]$ is a *set match* between D and P if there exists a string T in D such that T and P match on $[a..b]$, i.e., $T[a..b] = P[a..b]$ holds. Then we call the substring $P[a..b]$ *matched string*. We also say that a set match $[a..b]$ is a *set-right-maximal* match (SRMM) between D and P if we cannot extend the set match to the right, i.e., $b = n$ holds or $[a..b+1]$ is not set match between D and P. We also say that the set match $[a..b]$ is a *set-maximal match* (SMM) between D and P if the match cannot extend to both sides, i.e., the set match satisfies following two conditions: (1) $a = 1$ or $[a-1..b]$ is not a set match, and (2) $b = n$ or $[a..b+1]$ is not a set match. Figure 1 depicts SRMMs and SMMs for a dictionary $D = T_1, T_2, T_3, T_4$ such that $T_1 = 00000110$, $T_2 = 11011011$, $T_3 = 11110001$, and $T_4 = 00010010$, and pattern $P = 00111000$. The SRMMs between D and P are $[1..2], [2..2], [3..4], [4..6], [5..6], [6..7], [7..7]$ and $[8..8]$. The SMMs between D and P are $[1..2], [3..4], [4..6], [6..7]$ and $[8..8]$.

The SMM problem is to construct an index for a given dictionary D and threshold k supporting SMM queries. The SMM query for a given pattern P, denoted by $smm(D,P,k) = Y_1, Y_2, \dots, Y_n$, asks n SMMs starting at each position between D and P with threshold k. Formally, for a position $i \in [1,n]$, Y_i is the SMM starting at position i between D and P if the SMM exists and the length of the SMM is longer than k; otherwise Y_i is the empty interval $\emptyset$. For the example in Figure 1, $smm(D,P,2) = \emptyset, \emptyset, \emptyset, [4..6], \emptyset, \emptyset, \emptyset, \emptyset$.

For a dictionary D and an integer $i \in [1,n]$, let *i-suffixes*, denoted by $\mathit{Suffix}(D,i)$, be the set of suffixes starting at position i in the dictionary, i.e., $\mathit{Suffix}(D,i) = \{T_x[i..] \mid x \in [1,m]\}$. Similarly, let *$i$-substrings*, denoted by $\mathit{Substr}(D,i)$, be the set of substrings starting at position i in the dictionary and an empty string ε, i.e, $\mathit{Substr}(D,i) = \{T_x[i..j] \mid x \in [1,m], j \in [i,n]\} \cup \{\varepsilon\}$. Let $\mathit{Substr}(D,n+1) = \{\varepsilon\}$. For a string s, let the *i-text collection* for s, denoted by $set(D,i,s)$, be the set of texts in D such that each text T has s as the substring starting at position i, i.e., $set(D,i,s) = \{T_x \mid x \in [1,n], T_x[i..i+|s|-1] = s\}$. We call an i-substring s *i-substring of T* for a text $T \in set(D,i,s)$, respectively. For example, let D be the dictionary of Figure 1. Then $\mathit{Suffix}(D,4) = \{0001, 0010, 0110, 1011\}$ and $\mathit{Substr}(D,2) = \{\varepsilon, 0, 1, 01, 10, 11\}$.

Our computation model is a unit-cost word RAM with a machine word size of $\Omega(\log_2 nm)$ bits. We evaluate the space complexity in terms of the number of machine words. A bitwise evaluation of space complexity can be obtained with a $\log_2 nm$ multiplicative factor.

3. SMM automata

In this section, we present two automata outputting SMMs between a dictionary D and any pattern P with a threshold k. The former and latter SMM automata use $O(n^2 m\sigma)$ and $O(nm\sigma)$ space respectively, and hence we call those *naive* and *compact SMM automata*. First, we introduce two lemmas used in the following subsections.

▶ **Lemma 4.** *Let X_i and s_i be the i-th SRMM between D and P, and the matched string by X_i, respectively. (1) For*

Algorithm 1: An algorithm outputting the set of SMMs using SMM automaton.

```
Data: A dictionary $D$, pattern $P$, and threshold $k$.
Result: $smm(D, P, k) = Y_1, Y_2, \dots, Y_n$.
1 $v_{n+1} \leftarrow \varepsilon$;  /* $v_i$ is the node of the matched string by $X_i$ */
2 $\ell_{n+1} \leftarrow 0$;  /* $\ell_i$ is the length of $X_i$ */
3 for $i \leftarrow n; i \geq 1; i \leftarrow i - 1$ do
4     $v_i \leftarrow Z_i(v_{i+1}, P[i])$;  /* Compute $v_i$ using $i$-SMM sub-automaton */
5     $\ell_i \leftarrow$ the length of the string of $v_i$;
6     if $\ell_{i+1} \geq \ell_i$ and $\ell_{i+1} > k$ then output $Y_{i+1} \leftarrow [i + 1..i + \ell_{i+1}]$ ;
7     else output $Y_{i+1} \leftarrow \emptyset$ ;
8 if $\ell_1 > k$ then output $Y_1 \leftarrow [1..\ell_1]$;
9 else output $Y_1 \leftarrow \emptyset$ ;
```

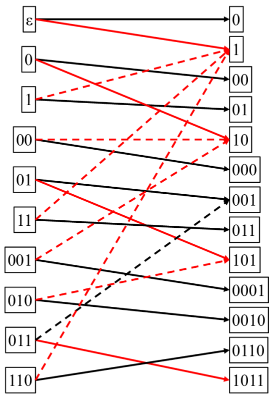

Figure 2 The 4-SMM sub-automaton for Figure 1. Black and red arrows are the edges labeled by the characters 0 and 1 respectively. Solid and dotted arrows are success and failure edges respectively. The left and right substrings are 3-substrings and 4-substrings in D.

an integer $i \in [2, n]$, a SRMM X_i is one of SMMs between D and P if and only if X_i is not shorter than the preceding SRMM X_{i-1} (i.e., $|X_i| \leq |X_{i-1}|$). (2) X_1 is one of SMMs between D and P. (3) For an integer $i \in [1, n - 1]$, if $X_i \neq \emptyset$, $s_i = P[i] \cdot K$ holds, where K is the $P[i]$-prefix of s_{i+1}; (4) If $|X_i| > |X_{i+1}|$ holds, $K = s_{i+1}$ holds (i.e., $s = P[i] \cdot s_{i+1}$); otherwise $|K| \leq |s_{i+1}|$ holds.

▶ **Lemma 5.** *For two i-substrings s and s' having the same text collection (i.e., $set(D, i, s) = set(D, i, s')$). For a character c, let q and q' be the c-next substrings of s and s' with boundary. Then the following statements hold: (1) q and q' have the same text collection, and (2) if $q \neq c \cdot s$ or $q' \neq c \cdot s'$ holds, $q = q'$ holds.*

3.1 The naive SMM automaton

3.1.1 Automaton based on Durbin's algorithm

We define the naive SMM automaton using Lemma 4. SMM automaton consists of n sub-automata and each i-th sub-automaton is called *i-SMM sub-automaton.* The i-th sub-automaton is a finite directed graph with labeled edges representing the relation between i-substrings and $(i+1)$-substrings on D. Each node corresponds a distinct i-substring or $(i + 1)$-substring in D. Nodes of $(i + 1)$-substrings and i-substrings only have outgoing and incoming edges, respectively. Therefore we call the former and latter nodes *input* and *output nodes*, respectively. We also call $(i + 1)$-substrings and i-substrings *input* and *output substrings*, respectively.

Every input node has σ outgoing edges with a distinct character. The edge labeled by a character c from an input node points to the output node of $c{\cdot}K$ if c is in i-substrings; otherwise it points to the output substring ε, where K is the c-prefix of the input substring. We call the output substring *c-output substring.* In addition, we call the edge *success edge* if the c-prefix of the input substring is equal to itself; otherwise we call it *failure edge.*

The number of the i-th sub-automaton is $|Substr(D, i)| + |Substr(D, i + 1)|$, and the number of edges is $\sigma|Substr(D, i + 1)|$. Since $|Substr(D, i)| \leq (n - i + 1)m \in O(nm)$ holds for $i \in [1, n]$, the number of edges and nodes in the naive SMM automaton is $O(n^2 m\sigma)$. Figure 2 depicts the 4-SMM sub-automaton for Figure 1. For example, the input node of the string 011 has the failure edge labeled by the character 0 and the success edge labeled by the character 1, and the failure edge points to the output node of the string 001 because output substrings do not contain the string 0011 but they contain the string 001.

3.1.2 Algorithm

We can compute SRMMs in the right-to-left order using the naive SMM automaton. Algorithm 1 is the pseudo code outputting SMMs with threshold k using SMM automaton. Let $Z_i(v, c)$ be the function outputting the node of the c-output substring of a given input node v in the naive i-SMM sub-automaton. In Line 4, the algorithm obtains the node of the matched string by X_i from the node of the matched string by X_{i+1} and the character $P[i]$ using the i-SMM sub-automaton Z_i for each integer $i \in [1, n]$. In Lines 6-7, the algorithm checks if X_{i+1} is one of SMMs and the length of X_{i+1} is larger than k using Lemma 4. Therefore, Algorithm 1 outputs $smm(D, P, k)$.

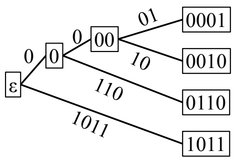

Figure 3 The compact trie for $F = \{0001, 0010, 0110, 1011\}$.

3.2 The compact SMM automaton

3.2.1 Automaton based on compact tries

We define the compact SMM automaton using Lemma 5 and *compact tries.* The compact trie for a set of strings is the trie for the set such that every internal node which has a single child is removed from the trie and the incoming and outcoming edges of the node are merged into a single edge, i.e., every internal node of the compact trie has at least two children. Each node represents the string by concatenating labels on the path from the root to the

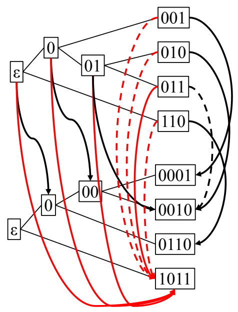

Figure 4 The compact 4-SMM sub-automaton for Figure 1. Black and red arrows are the edges labeled by the characters 0 and 1 respectively. Solid and dotted arrows are success and failure edges respectively. The upper and lower trees are the compact tries for 3-suffixes and 4-suffixes of the dictionary in Figure 1. We omit length labels of failure edges.

Algorithm 2: The modified main loop in Algorithm 1.

```
for $i \leftarrow n; i \geq 1; i \leftarrow i-1$ do
    /* $\hat{v}_i$ is the locus node of $X_i$ */
    $\hat{v}_i \leftarrow \hat{Z}_i(\hat{v}_{i+1}, P[i])$;   /* Compute $\hat{v}_i$ using compact $i$-SMM sub-automaton */
    if $\hat{Z}_i(\hat{v}_{i+1}, P[i])$ passes a failure edge then
        $\ell_i \leftarrow$ the length associated with the failure edge;
        if $\ell_{i+1} > k$ then output $Y_{i+1} \leftarrow [i+1..i+\ell_{i+1}]$ ;
        else output $Y_{i+1} \leftarrow \emptyset$ ;
    else
        $\ell_i \leftarrow \ell_{i+1} + 1$;
        output $Y_{i+1} \leftarrow \emptyset$;
```

node. We say that a node v is *the locus node* of a string s if v is the lowest node such that s is a prefix of the string represented by v. Figure 3 is the compact trie for the set of strings $F = \{0001, 0010, 0110, 1011\}$. For example, the locus node of 011 is the node 0110.

The compact SMM automaton consists of *compact* $1, 2, \ldots, n$*-SMM sub-automata* as the naive SMM automaton. Nodes of each $(i+1)$-SMM sub-automaton correspond to nodes in the compact tries for $(i+1)$-suffixes or i-suffixes. We call nodes for an $(i+1)$-suffix and i-suffix *input node* and *output node* respectively. Each input node has σ outgoing labeled edges with a distinct character to an output node in the i-th sub-automaton.

The edge labeled by a character c from the node points to the locus node of $c \cdot K$ in the compact trie for i-suffixes, which is also an output node in the sub-automaton, if c is in i-suffixes, where K is the c-prefix of the string by the input node in the naive i-SMM sub-automaton. Otherwise, the edge points to the root of the compact trie for i-suffixes. If the c-prefix of the input node is equal to the string of the input node, we call it *success edge*; otherwise we call it *failure edge*. In the compact SMM automaton, each failure edge stores the length of the string represented by the outcoming node. Figure 4 depicts the compact 4-SMM sub-automaton for Figure 1.

The number of nodes in each compact i-SMM sub-automaton is $O(m)$ because the number of nodes in the compact trie for i-suffixes is $O(m)$. Therefore, The number of nodes and edges in the compact SMM automaton is $O(nm\sigma)$.

3.2.2 Algorithm

Actually, if we keep a record of the length of X_i in addition to the state of the automaton, we can use the compact SMM automaton for the SMM problem, instead of the naive SMM automaton. We obtain the output by the compact i-SMM sub-automaton as follows. The input node X'_{i+1} has the outgoing edge labeled by the character $P[i]$ to an output node If the edge is success, the output is the output node and the integer $|X_{i+1}| + 1$; otherwise the output is the output node and the integer associated with the edge.

Algorithm 2 is the pseudo-code using compact SMM automaton instead of the main loop in Algorithm 1, and in the main loop, it computes the locus node X'_i using the compact i-SMM sub-automaton and the locus node X'_{i+1}. $\hat{Z}_i(v, c)$ is the function which returns the output node obtained by the input node v with a label c in the compact i-SMM sub-automaton. The algorithm computes SRMMs between the dictionary and pattern, and hence it can output SMMs with threshold k as Algorithm 1. Therefore we get Theorem 1.

4. Finding SMMs over FHE

In this section we show the way of solving the pp-SMM problem in the complexity stated in Theorem 3. To this end, we utilize fully homomorphic encryption (FHE) as a building block for secure computation. FHE is a form of encryption supporting addition and multiplication on ciphertexts. Some of FHE schemes particularly allow any party to evaluate an arbitrary Boolean circuit for a given input without the party learning the input and output. This implies that Boolean circuits outputting the SMMs for a given pattern can solve the pp-SMM problem in a single round. Combining this implication with Theorem 1, we finally produce such a circuit from the automaton in order to realize the efficient solution of the pp-SMM problem.

In the following subsections, we first explain the framework of FHE. Next, we introduce Lupanov representation, a way of representing Boolean function in circuit. For most of Boolean functions, this representation achieves the lower bound of the size of circuits equivalent to the functions. Then we show the outline of how we apply Lupanov representation to our case, describe the entire algorithm, and analyze its circuit-size complexity to prove Theorem 3. In the last two subsections, as a supplement, we refer to bootstrapping, which would be a concern in the actual implementation of our algorithm, and TFHE as one possible remedy for this concern. We omit the discussion about the formal protocol of our method and its security since those are trivial once we utilize FHE.

Algorithm 3: The modified main loop in Algorithm 2.

```
1 for $i \leftarrow n; i \geq 1; i \leftarrow i-1$ do
      /* $\tilde{v}_i$ is the locus node of $X_i$ represented in
         binary                                              */
2     $\tilde{v}_i \leftarrow \tilde{Z}_i(\tilde{v}_{i+1}, P[i])$;
3     $L_i \leftarrow W_i(\tilde{v}_{i+1}, P[i])$;
4     if $L_i \neq -1$ then $\ell_i \leftarrow L_i$;
5     else $\ell_i \leftarrow \ell_{i+1} + 1$;
6     if $L_i \neq -1$ and $\ell_i > k$ then output
       $Y_{i+1} \leftarrow [i+1..i+\ell_i]$;
7     else output $Y_{i+1} \leftarrow \emptyset$;
```

4.1 Fully homomorphic encryption (FHE)

FHE scheme (e.g., [12]) allows any party to execute addition and multiplication over ciphertexts, without the party learning the actual values of them.

FHE scheme consists of five algorithms (KeyGen, Enc, Dec, Add, Mult) working as follows:

KeyGen(λ) Return a secret key sk and public key pk using a given security parameter λ.

Enc(pk, π) Return the ciphertext ψ corresponding to $\pi \in \mathbb{Z}_q$ under a given public key pk.

Dec(sk, ψ) Return the plaintext π recovered from a ciphertext ψ using the secret key sk, i.e., $\mathsf{Dec}(sk, \mathsf{Enc}(pk, \pi)) = \pi$ holds.

Add(pk, ψ_1, ψ_2) Return the ciphertext ψ such that $\mathsf{Dec}(sk, \psi) = \mathsf{Dec}(sk, \psi_1) + \mathsf{Dec}(sk, \psi_2)$ holds.

Mult(pk, ψ_1, ψ_2) Return the ciphertext ψ such that $\mathsf{Dec}(sk, \psi) = \mathsf{Dec}(sk, \psi_1) \times \mathsf{Dec}(sk, \psi_2)$ holds.

Particularly, when $q = 2$ (i.e., the plaintext space is restricted to 1 bit), Add(pk, ψ_1, ψ_2) and Mult(pk, ψ_1, ψ_2) correspond to the bitwise XOR operation and the bitwise AND operation, respectively. Since it is known that those operations are sufficient to express any Boolean function, in this case FHE schemes permit evaluating an arbitrary Boolean circuit, keeping the input and output encrypted. This fact implies that we can achieve an algorithm for the pp-SMM problem by simply translating into a circuit plain-text algorithms of the SMM problem. Among the algorithms of the SMM problem, particularly our plaintext algorithm, Algorithm 2 is suitable for this purpose because this algorithm is as fast as Durbin's algorithm, and "deamortized".

4.2 Circuit representation of the algorithm

4.2.1 Outline of our representation and analysis of complexity

In order to achieve a smallcircuit, we leverage the fact that any logical function $f : \{0,1\}^r \to \{0,1\}$ can be represented as a Boolean circuit of size $O(\frac{2^r}{r})$ for an integer r by *Lupanov representation* [24], and hence any mapping $f : \{0,1\}^r \to \{0,1\}^t$ can be computed with a Boolean circuit of size $O(\frac{2^r}{r}t)$ for two integers r and t.

The key idea for efficiently utilizing Lupanov representation is that the transition induced by each character input is a mapping $\hat{Z}_i : \{1, O(m)\} \times \{1, \sigma\} \to \{1, O(m)\}$ when we number the states of the compact sub-automata from 1 to $O(m)$. By analogy with binary representation, one can see this could be regarded as $\tilde{Z}_i : \{0,1\}^{O(\log(m\sigma))} \to \{0,1\}^{O(\log m)}$. Then it immediately follows that each transition can be described with $O(\frac{m\sigma}{\log(m\sigma)} \log m)$ circuit elements. In total, this results in $O(nm\sigma \frac{\log m}{\log(m\sigma)})$ circuit elements for expressing the whole transitions. Likewise, the conditional branch in Algorithm 2 can be realized in size $O(nm\sigma \frac{\log n}{\log m\sigma})$.

The entire procedures of our algorithm are shown in Algorithm 3. One can see that the bottleneck of the complexity should be the parts where Lupanov representation is used. Thus, in total, Algorithm 3 can be described in the circuit of size $O(\frac{nm\sigma \log(nm)}{\log m\sigma})$. Splitting input characters with binary representation, we can decrease the complexity to $O(\frac{nm \log \sigma \log(nm \log \sigma)}{\log m})$, and obtain Theorem 3.

4.2.2 Bootstrapping

For the purpose of ensuring security, FHE scheme usually embeds noise in its ciphertexts. The noise grows each time of applying homomorphic operations, until ultimately it becomes a non-negligible amount to correctly decrypt the ciphertext. Therefore, such FHE schemes are originally unable to perform an infinite number of homomorphic operations, especially multiplication. In brief, bootstrapping is a method to reduce such noise.

Bootstrapping is an indispensable procedure for the unlimited use of homomorphic operations, but at the same time the procedure is notorious for its heavy computational cost. Due to this disadvantage, formerly FHE was usually considered to be unusable for algorithms which require a large number of homomorphic operations. Our algorithm is no exception because the number of bootstrappings required in our circuit is in proportion to the length of strings.

4.2.3 TFHE

In order to overcome the difficulty of bootstrapping, we especially adopt TFHE [9] as the underlying foundation in the implementation of our algorithm. TFHE is a FHE scheme, and it also indicates the library based on the scheme, which provides addition and multiplication over ciphertexts of $\mathbb{Z}_2$. TFHE is known for its fast bootstrapping and already has various applications [5, 28, 8].

We note that our algorithm can be implemented also with other FHE schemes than TFHE as long as they enable us to evaluate Boolean circuits of any depth. Therefore, if the improvement of efficiency in such FHE schemes is made, then our algorithm can immediately enjoy the improvement.

5. Experiments

In this section, we show the performance of our algorithm by measuring its actual run time. For this purpose, we carried out two experiments of comparing our algorithm with the existing one on the randomly produced dataset and on actual genome data.

5.1 Setups

5.1.1 Datasets

In the first experiment, we would like to show the difference of complexities between ours and the existing method proposed by Shimizu et al. [30]. To this end, as a dataset, we fixed the number of strings as 1000, and randomly generated strings with Mersenne Twister, varying the length of strings from 300 to 1500.

We did not employ actual genome data as the dataset of the first experiment. In order to use genome data in this experiment, we needed to cut out the haplotype sequences in the data, to obtain the substrings of the desired lengths. However, genome data has bias in its values and the run times could be much smaller than on random datasets and dependent on the range in which we cut out the strings. This may result in an inaccurate result, where we cannot see the correct aymptotic proportions of the run times. For this reason, we decided that we did not employ genome data in the first experiment.

Since the first experiment was performed with the random dataset, we need to show our method runs faster than the method by Shimizu et al. even on genome data, given that the length of strings is large enough. Therefore, in contrast, the second experiment was carried out with genome data brought from the 1000 Genomes Project phase 1 data release [10]. However, because the raw genome data is too large to complete the measurement, helplessly we choose 1000 haplotype sequences, randomly choose a range, and cut out the haplotype sequences in the data in that range, in order to obtain the substrings of length 1000 and of length 1200. We assume that we achieved the goal of this experiment with the result of those lengths.

5.1.2 Implementation and environment

We implemented the proposed algorithm in C++ with the open source library of TFHE [2]. The parameter of TFHE λ was set to provide at least 128-bit security level. At the stage of converting a given dictionary into circuits, we also utilized Succinct Data Structure Library [1] to implement PBWT and compact trie. Our implementation is available on `https://github.com/ykoike-MDL/NISMM`.

We compared our method with the method proposed by Shimizu et al. . Their method is based on oblivious transfer, and requires communications between server and client. For a fair comparison, we launched their server program and client program locally in order to ignore their communication cost. We measured the "end-to-end" run times of both methods with a high resolution clock (i.e., the standard chrono library). In other words, we considered not only the run time of server program, but also the one of client program.

We performed experiments on a 6-core machine with a 3.70 GHz Intel Core i7-8700 processor and 32 GB memory. All the programs were compiled by gcc-7.4.0 with the

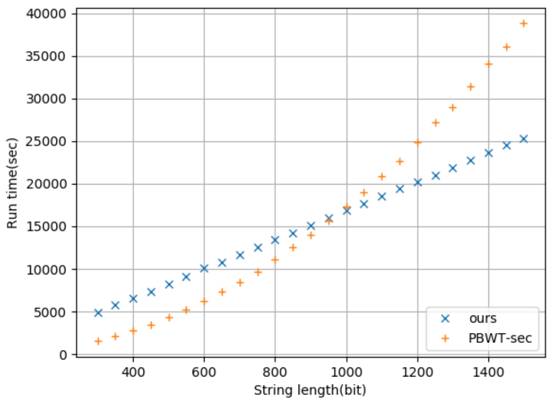

Figure 5 The plot of run time of our method and the existing one (PBWT-sec) on 1000 strings.

optimization flag -O3, and run on Ubuntu 18.04. We also parallelized all the programs to make them use 4 threads.

5.2 Results

The result of the first experiment is shown in Figure 5. The observed run time of ours is almost linear in the length of strings, while the method by Shimizu et al. shows quadratic growth. This result fits the theoretical complexity described in Table 1. Since the run times of both methods are well proportional to their theoretical complexity, it is possible to estimate their run time even in the case where the strings of greater lengths are given. When the lengths are 10^4 and 10^5, then at the most our method would take 3 days and 1 month, respectively. Thus, our method would report answers in a realistic period of time even for larger datasets which are often provided as actual genome data.

Table 2 shows the result of the second experiment. For the sake of comparison, we put on the same table the run times of both methods on the random dataset. As described in Section 5.1.1, there are noticeable differences between the run time of ours on the random dataset and the one on the genome dataset. This is attributed to the size of the resultant PBWT and SMM automaton. Due to the bias of values of the genome data, the sizes of those data structures become relatively small. PBWT-sec, however, shows little difference. In fact, in their method, the size of PBWT is not likely to influence the run time much. This result would suggest that, on genome data, our method is expected to run faster than its average performance estimated by the result on the random dataset.

Moreover, we note that the run time of our method can be improved by reducing the size of circuits since the run time of our method is directly affected by the size of the circuit we evaluate. Lupanov representation is the method of producing circuits which achieves the lower bound only in the worst cases, not in every case (i.e, Lupanov representation is not a minimization method). Therefore, if we adopt another method which produces smaller circuits

Table 2 Running times of ours and the existing method (PBWT-sec) on two datasets, with the concrete size of the circuits ours produced.

	PBWT-sec(sec)	Ours(sec)	Ours(number of circuit elements)
Genome (length = 1000)	17290.7	3651.27	1994378
Genome (length = 1200)	24960.7	4571.59	2443029
Random (length = 1000)	17282.90	16845.62	9032524
Random (length = 1200)	24873.81	20199.96	10855608

than Lupanov representation, we can immediately see the improvement of the performance, probably at the expense of efficiency in preprocessing.

References

[1] simongog/sdsl: Succinct data structure library. `https://github.com/simongog/sdsl`. (Accessed on 08/01/2019).

[2] Tfhe fast fully homomorphic encryption over the torus. `https://github.com/tfhe/tfhe`. (Accessed on 08/01/2019).

[3] P. R. Billings, M. A. Kohn, M. de Cuevas, J. Beckwith, J. S. Alper, and M. R. Natowicz. Discrimination as a consequence of genetic testing. *American journal of human genetics*, 50(3):476–482, Mar 1992. 1539589[pmid].

[4] Marina Blanton and Mehrdad Aliasgari. Secure outsourcing of DNA searching via finite automata. In Sara Foresti and Sushil Jajodia, editors, *Data and Applications Security and Privacy XXIV, 24th Annual IFIP WG 11.3 Working Conference, Rome, Italy, June 21-23, 2010. Proceedings*, volume 6166 of *Lecture Notes in Computer Science*, pages 49–64. Springer, 2010.

[5] Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. Fast homomorphic evaluation of deep discretized neural networks. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part III*, volume 10993 of *Lecture Notes in Computer Science*, pages 483–512. Springer, 2018.

[6] Sharon R. Browning and Brian L. Browning. Identity-by-descent-based heritability analysis in the northern finland birth cohort. *Human genetics*, 132(2):129–138, Feb 2013. 23052944[pmid].

[7] Christopher L. et al. Campbell. North african jewish and non-jewish populations form distinctive, orthogonal clusters. *Proceedings of the National Academy of Sciences*, 109(34):13865–13870, 2012.

[8] Sergiu Carpov, Nicolas Gama, Mariya Georgieva, and Juan Ramón Troncoso-Pastoriza. Privacy-preserving semi-parallel logistic regression training with fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2019:101, 2019.

[9] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, pages 3–33, 2016.

[10] The 1000 Genomes Project Consortium. An integrated map of genetic variation from 1,092 human genomes. *Nature*, 491:56 EP –, Oct 2012. Article.

[11] Richard Durbin. Efficient haplotype matching and storage using the positional burrows-wheeler transform (PBWT). *Bioinformatics*, 30(9):1266–1272, 2014.

[12] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of STOC*, pages 169–178, 2009.

[13] Melissa Gymrek, Amy L McGuire, David Golan, Eran Halperin, and Yaniv Erlich. Identifying personal genomes by surname inference. *Science (New York, N.Y.)*, 339:321–324, 01 2013.

[14] David J. et al. Hunter. A genome-wide association study identifies alleles in fgfr2 associated with risk of sporadic post-menopausal breast cancer. *Nature genetics*, 39(7):870–874, Jul 2007. 17529973[pmid].

[15] Liina Kamm, Dan Bogdanov, Sven Laur, and Jaak Vilo. A new way to protect privacy in large-scale genome-wide association studies. *Bioinformatics*, 29(7):886–893, 02 2013.

[16] Marty et al. Kardos. Inferring individual inbreeding and demographic history from segments of identity by descent in ficedula flycatcher genome sequences. *Genetics*, 205(3):1319–1334, 2017.

[17] Augustine et al. Kong. Detection of sharing by descent, long-range phasing and haplotype imputation. *Nature Genetics*, 40:1068 EP –, Aug 2008. Article.

[18] B Langmead, C Trapnell, M Pop, and S.L. Salzberg. Ultrafast and memory-efficient alignment of short dna sequences to the human genome. *Genome Biol*, 10, 01 2009.

[19] Ruiqiang et al. Li. Soap2: An improved ultrafast tool for short read alignment. *Bioinformatics (Oxford, England)*, 25:1966–7, 07 2009.

[20] Zhen Lin, Art B. Owen, and Russ B. Altman. Genomic research and human subject privacy. *Science*, 305(5681):183–183, 2004.

[21] Oren E. et al. Livne. Primal: Fast and accurate pedigree-based imputation from sequence data in a founder population. *PLOS Computational Biology*, 11(3):1–14, 03 2015.

[22] Wenjie Lu, Shohei Kawasaki, and Jun Sakuma. Using fully homomorphic encryption for statistical analysis of categorical, ordinal and numerical data. In *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*. The Internet Society, 2017.

[23] Wenjie Lu, Yoshiji Yamada, and Jun Sakuma. Efficient secure outsourcing of genome-wide association studies. In *Proceedings of the 2015 IEEE Security and Privacy Workshops*, SPW '15, pages 3–6, Washington, DC, USA, 2015. IEEE Computer Society.

[24] O. B. Lupanov. A method of circuit synthesis. *Izvesitya VUZ*, 1:120–140, 1958.

[25] Michael Naehrig, Kristin E. Lauter, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In Christian Cachin and Thomas Ristenpart, editors, *Proceedings of the 3rd ACM Cloud Computing Security Workshop, CCSW 2011, Chicago, IL, USA, October 21, 2011*, pages 113–124. ACM, 2011.

[26] Alkes L. et al. Price. Single-tissue and cross-tissue heritability of gene expression via identity-by-descent in related or unrelated individuals. *PLoS genetics*, 7(2):e1001317–e1001317, Feb 2011. 21383966[pmid].

[27] MO Rabin. How to exchange secrets by oblivious transfer, report no. Technical report, TR-81, Harvard Aiken Computation Laboratory, 1981.

[28] Amartya Sanyal, Matt J. Kusner, Adrià Gascón, and Varun Kanade. TAPAS: tricks to accelerate (encrypted) prediction as a service. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 4497–4506. PMLR, 2018.

[29] Hirohito Sasakawa, Hiroki Harada, David duVerle, Hiroki Arimura, Koji Tsuda, and Jun Sakuma. Oblivious evaluation of non-deterministic finite automata with application to privacy-preserving virus genome detection. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, WPES '14, pages 21–30, New York, NY, USA, 2014. ACM.

[30] Kana Shimizu, Koji Nuida, and Gunnar Rätsch. Efficient privacy-preserving string search and an application in genomics. *Bioinformatics*, 32(11):1652–1661, 2016.

[31] Xiaowen et al. Sun. Slaf-seq: An efficient method of large-scale de novo snp discovery and genotyping using high-throughput sequencing. *PLOS ONE*, 8(3):1–9, 03 2013.

[32] Oleksandr Tkachenko, Christian Weinert, Thomas Schneider, and Kay Hamacher. Large-scale privacy-preserving statistical computations for distributed genome-wide association studies. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, ASIACCS '18, pages 221–235, New York, NY, USA, 2018. ACM.

[33] D. Wu and J. Haven. Using homomorphic encryption for large scale statistical analysis. `https://www.cs.virginia.edu/dwu4/posters/FHE-SI.pdf`, 2012. (Accessed on 06/09/2019).