Suffix Array of Alignment: A Practical Index for Similar Data

Joong Chae Na[1], Heejin Park[2], Sunho Lee[3], Minsung Hong[3], Thierry Lecroq[4], Laurent Mouchard[4], and Kunsoo Park[3,⋆]

[1] Department of Computer Science and Engineering, Sejong University, Korea
jcna@sejong.ac.kr
[2] College of Information and Communications, Hanyang University, Korea
hjpark@hanyang.ac.kr
[3] School of Computer Science and Engineering, Seoul National University, Korea
{slee,mshong,kpark}@theory.snu.ac.kr
[4] Department of Computer Science, University of Rouen, France
{Thierry.Lecroq,Laurent.Mouchard}@univ-rouen.fr

Abstract. The *suffix tree of alignment* is an index data structure for similar strings. Given an alignment of similar strings, it stores all suffixes of the alignment, called *alignment-suffixes*. An alignment-suffix represents one suffix of a string or suffixes of multiple strings starting at the same position in the alignment. The suffix tree of alignment makes good use of similarity in strings theoretically. However, suffix trees are not widely used in biological applications because of their huge space requirements, and instead suffix arrays are used in practice.
In this paper we propose a space-economical version of the suffix tree of alignment, named the *suffix array of alignment (SAA)*. Given an alignment ρ of similar strings, the SAA for ρ is a lexicographically sorted list of all the alignment-suffixes of ρ. The SAA supports pattern search as efficiently as the *generalized suffix array*. Our experiments show that our index uses only 14% of the space used by the generalized suffix array to index 11 human genome sequences. The space efficiency of our index increases as the number of the genome sequences increases. We also present an efficient algorithm for constructing the SAA.

Keywords: Indexes for similar data, suffix arrays, alignments

1 Introduction

The 1000 Genomes project [4] is aiming at building a database of 1092 individual human genome sequences using a cheap and fast sequencing, called Next Generation Sequencing (NGS). To sequence an individual genome using the NGS, the individual genome is divided into short segments (called reads) and they are aligned to the Human reference Genome. This is possible because an individual

⋆ Corresponding author.

genome is more than 99% identical to the Human reference Genome. The similarity also enables us to store individual genomes efficiently. Instead of storing 1000 whole individual sequences, only 1% different regions of each individual genome can be stored.

Not only efficient storing techniques but also efficient indexing techniques for similar strings have been developed. The first such index was proposed by Mäkinen et al. [16, 17]. Their index uses run-length encoding, a suffix array, and BWT [3]. Huang et al. [9] indexed similar strings by building separate data structures for common regions and non-common regions. In addition, indexes based on Lemple-Ziv compression schemes [15, 21] have been developed [5, 14]. Some of these indexes are surveyed in [20]. The space reductions of these indexes are achieved mostly by using classical compressed indexes. However, the indexes do not support efficient pattern search or require auxiliary data structures to improve the pattern search time.

Recently, a suffix tree for similar strings, called a *suffix tree of alignment* [19], have been proposed without sacrificing the pattern search time, i.e., the suffix tree of alignment supports linear-time pattern search. Given an alignment of similar strings, the suffix tree of alignment stores suffixes of an alignment, called *alignment-suffixes* (for short a-suffixes) rather than suffixes of a string. An a-suffix may represent suffixes of multiple strings starting at the same position in an alignment. The suffix tree of alignment makes good use of similarity in strings theoretically. Although suffix trees support many functionalities [2, 8], however, they are not widely used in biological applications because of the huge space requirement. Instead, suffix arrays [18] (including their compressed forms [6, 7]) are widely used in practice.

In this paper we propose the *suffix array of alignment (SAA)*, an array version of the suffix tree of alignment. Given an alignment ρ, the SAA for ρ is a lexicographically sorted list of all the a-suffixes of ρ. We show that the sorted order of the a-suffixes is well defined and the longest common prefix (lcp) of two a-suffixes is also well defined. Assume that given strings consist of common regions and non-common regions alternatively, e.g., three strings A, B, and C can be represented as $A = \alpha_1\beta_1 \dots \alpha_k\beta_k\alpha_{k+1}$, $B = \alpha_1\delta_1 \dots \alpha_k\delta_k\alpha_{k+1}$, and $C = \alpha_1\vartheta_1 \dots \alpha_k\vartheta_k\alpha_{k+1}$, where α_i's are common regions and β_i's, δ_i's and ϑ_i's are non-common regions. Then, the SAA requires $O(|A|+\sum_{i=1}^{k}(2|\alpha_i^*|+|\delta_i|+|\vartheta_i|))$ space, where α_i^* is the longest suffix of α_i appearing at least twice in A, in B or in C. (For simplicity, three strings are considered but our results work well for more than three strings.) The space requirement of the SAA is asymptotically the same as that of the suffix tree of alignment, but the SAA is more space-efficient practically. Furthermore, our suffix array supports pattern search as efficiently as the generalized suffix array (GSA).

Moreover, we show by experiments that our index is space-efficient for similar data in practice by analyzing and comparing the space requirements of the SAA and the GSA, which support the same efficiency of pattern search. The space requirement of our index is influenced by the lengths of α_i^* and non-common regions. Our experiments show that these lengths are short in practice and thus

our index consumes very small space. We used 11 human genome sequences, one reference sequence and 10 individual sequences from the 1000 Genomes project website[5]. In the genome sequences, non-common regions are only 0.3% of the entire positions, i.e., these sequences are very similar. Moreover, the α_i^*'s, which is a main factor for the space requirement of the SAA, occupy 5% of the entire positions and the length of α_i^* is 16.64 on average. Conclusively, the SAA requires only 14% of the space required by the GSA for indexing the 11 sequences. It should be noted that the space efficiency of our index increases as the number of the genome sequences increases.

We also present an efficient algorithm for constructing the SAA. One might think the SAA can be simply constructed by simulating the algorithm for constructing the suffix tree of alignment in [19]. However, it is not easy because the algorithm heavily uses the dynamic property of the suffix tree and makes use of suffix links. The core of the tree construction algorithm is how to compute α_i^* efficiently, which is solved using a property satisfied in a partial suffix tree containing suffixes derived from several strings. Thus, we developed a new algorithm to compute α_i^* using only suffix arrays. For this, we generalize the property dedicated to the suffix tree so that the property is satisfied in substrings of input strings. Conclusively, we can compute α_i^* and thus construct the SAA as efficiently as the algorithm in [19].

2 Suffix array of alignment (SAA)

In this section we define the suffix array of alignment (SAA) and present how to construct the SAA. For simplicity, we consider only alignments of three strings but our definitions and algorithms can be easily extended to more than three strings. We first consider alignments with one non-common chunk and then general alignments with more than one non-common chunk.

2.1 Definition of SAA

Let A, B, and C be similar strings such that $A = \alpha\beta\gamma$, $B = \alpha\delta\gamma$, and $C = \alpha\vartheta\gamma$, where α and γ are common regions in all strings, and β, δ, and ϑ are non-common regions. Then, these regions represent an alignment of the strings and each string can be transformed to another string by replacing non-common regions. We denote this alignment of the three strings by $\rho = \alpha(\beta/\,\delta/\,\vartheta)\gamma$. For simplicity, we assume that all strings end with a special symbol $\# \in \Sigma$ occurring nowhere else in the strings.

The suffixes of the alignment ρ, called *alignment-suffixes* (for short *a-suffixes*), are defined as in [19]. Let α^a, α^b, and α^c be the longest suffixes of α occurring at least twice in the strings A, B, and C, respectively. Let α^* be the longest of α^a, α^b, and α^c, i.e., α^* is the longest suffix of α occurring at least twice in A, in B, or in C. Then, these are a-suffixes of ρ, which are classified into 5 types.

[5] http://www.1000genomes.org/

idx	*POS*	*LCP*	a-suffixes (type)
1	(1, 9)	-	`#` (1)
2	(2, 6)	0	`a`$\underline{\texttt{a}}$`ab#` (3)
3	(1, 4)	3	`a`$\underline{\texttt{ab}}$`ab#` (2)
4	(3, 4)	2	`a`$\underline{\texttt{ac}}$`ab#` (4)
5	(1, 7)	1	`ab#` (1)
6	(2, 4)	2	`a`$\underline{\texttt{ba}}$`ab#` (3)
7	(1, 5)	3	$\underline{\texttt{ab}}$`ab#` (2)
8	(1, 1)	2	`abca(`$\underline{\texttt{ab/ba/ac}}$`)ab#` (5)
9	(3, 5)	1	$\underline{\texttt{ac}}$`ab#` (4)
⋮	⋮	⋮	⋮

Fig. 1. The SAA of `abca(`$\underline{\texttt{ab/ba/ac}}$`)ab#`. A pair (a, b) in POS represents the string number a and the starting position b of an a-suffix. $LCP[i]$ is the length of lcp between two a-suffixes of $POS[i-1]$ and $POS[i]$.

1. a suffix of γ,
2. $\omega^a\gamma$, where ω^a is a (non-empty) suffix of $\alpha^*\beta$.
3. $\omega^b\gamma$, where ω^b is a (non-empty) suffix of $\alpha^*\delta$.
4. $\omega^c\gamma$, where ω^c is a (non-empty) suffix of $\alpha^*\vartheta$.
5. $\alpha'(\beta/\,\delta/\,\vartheta)\gamma$, where α' is a suffix of α longer than α^*.

For example, assume that an alignment `abca(`$\underline{\texttt{ab/ba/ac}}$`)ab#` is given. Then, $\alpha^a = \alpha^b =$ `a` and $\alpha^c =$ `ca`. Since α^* is `ca`, `ca`$\underline{\texttt{ab}}$`ab#` is an a-suffix of type 2 and `bca(`$\underline{\texttt{ab/ba/ac}}$`)ab#` is an a-suffix of type 5.

The *suffix array of alignment (SAA)* for ρ is a lexicographically sorted list of all the a-suffixes of ρ. It is clear what the sorted order for a-suffixes of types 1-4 is since an a-suffix of types 1-4 represents one string. On the other hand, an a-suffix of type 5, e.g., $\omega = \alpha'(\beta/\,\delta/\,\vartheta)\gamma$ where $|\alpha'| > |\alpha^*|$ represents three strings $\alpha'\beta\gamma$, $\alpha'\delta\gamma$, and $\alpha'\vartheta\gamma$ derived from A, B, and C, respectively. However, it does not cause trouble when determining the order of ω between the a-suffixes of ρ. Since α' occurs only once in each string, i.e, as prefix of $\alpha'\beta\gamma$, $\alpha'\delta\gamma$, and $\alpha'\vartheta\gamma$, the order of ω is determined by α'. Thus, the lexicographically sorted order between the a-suffixes is well defined and the longest common prefix (lcp), an additional information often used together with the suffix arrays, between a-suffixes of ρ is also well defined. See Figure 1 for an example.

The space requirement of the SAA is linear to the number of a-suffixes. There are $|\gamma|$ a-suffixes of type 1. The number of a-suffixes of types 2, 3, and 4 is $|\alpha^*\beta| + |\alpha^*\delta| + |\alpha^*\vartheta|$) and the number of a-suffixes of type 5 is $|\alpha| - |\alpha^*|$. Since $|A| = |\alpha| + |\beta| + |\gamma|$, the SAA of ρ requires $O(|A| + 2|\alpha^*| + |\delta| + |\vartheta|)$ space.

2.2 Construction of SAA

One method for constructing the SAA of ρ is using the suffix tree of the alignment ρ [19] as an intermediate index. However, this method does not make full use of the space-efficiency of suffix arrays because suffix trees require much more

space than suffix arrays. Another method, without constructing suffix trees, is constructing first the generalized suffix array (GSA) for the three strings as an intermediate index and then deleting suffixes that are not a-suffixes in the GSA. However, this method also is not efficient in working space as well as in construction time because the time and space requirement of the GSA is proportional to the total length of the strings regardless of similarity among the strings. The more number of strings are in the alignment and the more similar the strings are, the more is the inefficiency.

We present how to construct the SAA efficiently in time and space. Our algorithm for constructing the SAA of ρ consists of three steps. Let γ^a, γ^b, and γ^c be the longest prefixes of γ occurring at least twice in the strings A, B, and C, respectively. Let γ^* be the longest of γ^a, γ^b, and γ^c. (Note that these definitions are symmetrical with those of α^a, α^b, α^c, and α^*, and are different from the definition of $\hat{\gamma}$ used in [19].) Then, the outline of our algorithm is as follows:

1. Compute $|\alpha^*|$ and $|\gamma^*|$.
2. Construct the GSA for three strings A, $\alpha^*\delta\gamma^*d$, and $\alpha^*\vartheta\gamma^*d$, where d is the symbol following γ^* in γ.
3. Delete suffixes of γ^*d derived from $\alpha^*\delta\gamma^*d$ and $\alpha^*\vartheta\gamma^*d$.

Step 1 is the core step of our algorithm. We mainly focus on the problem of computing $|\alpha^*|$ since $|\gamma^*|$ can be computed symmetrically. For a string S, let S^R be the reversed string of S. We can compute $|\alpha^a|$ by searching for α^R in the suffix array of A^R. Thus, one method to compute $|\alpha^*|$ is constructing the suffix array of each reversed string and computing $|\alpha^a|$, $|\alpha^b|$, and $|\alpha^c|$. However, this method requires the time proportional to the total length of the three strings due to constructing the three suffix arrays.

To compute $|\alpha^*|$ more efficiently, we make use of the similarity in the strings. Consider the strings A and B. The following lemma says that, given $|\alpha^a|$, a substring including δ is sufficient for computing $\max(|\alpha^a|, |\alpha^b|)$ instead of the entire of B. (Note that we do not need to compute the exact value of $|\alpha^b|$ to compute $|\alpha^*|$.)

Lemma 1. *If $|\alpha^b| > |\alpha^a|$, α^b occurs in the substring B' of B, where $B' = \alpha^a\delta\gamma^a$.*

Proof. By definition of α^b, there are at least two occurrences of α^b in B. Obviously, one occurrence occ_1 of α^b appears as a suffix of α. Since $|\alpha^b| > |\alpha^a|$, occ_1 cannot be included in B'. Let occ_2 denote an occurrence of α^b in B other than occ_1. Let s_2 and e_2 be the starting and the ending positions of occ_2 in B, respectively. Let s_a and e_a be the starting and the ending positions of the substring B' in B, respectively.

We show that occ_2 is included in B', i.e., $s_a \leq s_2$ and $e_2 \leq e_a$. We first prove by contradiction that $s_a \leq s_2$. Suppose $s_2 < s_a$. We have two cases according to whether occ_2 is overlapped with δ or not.

– The case when occ_2 is not overlapped with δ. Then, occ_2 is included in α and it means that there are at least two occurrences (occ_1 and occ_2) of α^b in α and also in A. It contradicts with the definition of α^a since $|\alpha^b| > |\alpha^a|$.

– The case when occ_2 is overlapped with δ. Let α' be the suffix of α starting at s_2. Since $s_2 < s_a$, $|\alpha'| > |\alpha^a|$. Since α' is a prefix of occ_2, α' is also a prefix of occ_1. Hence, there are at least two occurrences of α' in α and also in A. It contradicts with the definition of α^a since $|\alpha'| > |\alpha^a|$.

Similarly, we can prove that $e_2 \leq e_a$ by contradiction with the definition of γ^a. □

Note that this property also holds for other strings. For example, if $|\alpha^c| > |\alpha^a|$, α^c occurs in the substring $\alpha^a\vartheta\gamma^a$ of C.

Using this property, we can compute $|\alpha^*|$ and $|\gamma^*|$ as follows:

1.1 Compute $|\alpha^a|$ by searching for α^R in the suffix array of A^R and, symmetrically, compute $|\gamma^a|$ by searching for γ in the suffix array of A.

1.2 Compute $\ell^b = \max(|\alpha^a|, |\alpha^b|)$ using the suffix array of $(\alpha^a\delta\gamma^a)^R$ as follows. Let α' be the longest suffix of α occurring in $\alpha^a\delta\gamma^a$. (Note that $|\alpha'| \geq |\alpha^a|$ since α^a occurs in $\alpha^a\delta\gamma^a$.) We can find α' by searching for α^R in the suffix array of $(\alpha^a\delta\gamma^a)^R$. By Lemma 1, if $|\alpha'| > |\alpha^a|$, $\ell^b = |\alpha'|$ and, otherwise, $\ell^b = |\alpha^a|$.
Symmetrically, compute $\max(|\gamma^a|, |\gamma^b|)$ using the suffix array of $\alpha^a\delta\gamma^a$.

1.3 Similarly, compute $\ell^c = \max(|\alpha^a|, |\alpha^c|)$ using the suffix array of $(\alpha^a\vartheta\gamma^a)^R$. Then, $|\alpha^*| = \max(\ell^b, \ell^c)$.
Symmetrically, compute $\max(|\gamma^a|, |\gamma^c|)$ and $|\gamma^*|$.

Since the suffix array of a string S can be constructed using $O(|S|)$ time and $O(|S|)$ space [10, 12, 13], and one can search for a string P using the suffix array of S with some auxiliary information in $O(|P|)$ [1, 11], computing $|\alpha^*|$ and $|\gamma^*|$ requires $O(|A| + |\alpha^*\delta\gamma^*| + |\alpha^*\vartheta\gamma^*|)$ time and $O(|A|)$ working space. Note that the suffix array constructed in each substep is needed only in the substep.

In Step 2, we construct the GSA for three strings A, $\alpha^*\delta\gamma^*d$, and $\alpha^*\vartheta\gamma^*d$, where d is the symbol following γ^* in γ. The GSA contains all the a-suffixes of the alignment ρ. (Note that the suffixes of γ in A are the a-suffixes of type 1 of ρ and the suffixes of A longer than $\alpha^*\beta\gamma$ can be *implicitly* converted to the a-suffixes of type 5 of ρ [19].) The reason why γ^*d is necessary is as follows. Let $\omega\gamma$ be an a-suffix of type 3 (of B). To determine the order of $\omega\gamma$ among a-suffixes of ρ, we may need a prefix of γ. Since γ^*d occurs only once in each string, the order of $\omega\gamma$ is determined by $\omega\gamma^*d$. Obviously, Step 2 requires $O(|A|+|\alpha^*\delta\gamma^*|+|\alpha^*\vartheta\gamma^*|)$ time and space.

In Step 3, we delete suffixes of γ^*d in $\alpha^*\delta\gamma^*d$ and $\alpha^*\vartheta\gamma^*d$ because these are redundant with suffixes of A (a-suffixes of type 1). Consider a suffix ω of γ^*d. In the GSA, there are two ω's derived from $\alpha^*\delta\gamma^*d$ and $\alpha^*\vartheta\gamma^*d$. The two ω's are adjacent in the GSA. We can delete redundant suffixes by scanning the entire GSA, which requires $O(|A| + |\alpha^*\delta\gamma^*| + |\alpha^*\vartheta\gamma^*|)$ time and space.

Theorem 1. *Given an alignment $\rho = \alpha(\beta/\,\delta/\,\vartheta)\gamma$, the SAA of ρ can be constructed using $O(|A| + |\alpha^*\delta\gamma^*| + |\alpha^*\vartheta\gamma^*|)$ time and working space.*

2.3 Alignment with multiple regions

In this section we consider alignments with multiple non-common regions. Let $A = \alpha_1\beta_1 \dots \alpha_k\beta_k\alpha_{k+1}$, $B = \alpha_1\delta_1 \dots \alpha_k\delta_k\alpha_{k+1}$, and $C = \alpha_1\vartheta_1 \dots \alpha_k\vartheta_k\alpha_{k+1}$. We denote the alignment of the strings by $\rho = \alpha_1(\beta_1/\,\delta_1/\,\vartheta_1)\alpha_2(\beta_2/\,\delta_2/\,\vartheta_2)\alpha_3 \dots \alpha_{k+1}$. Without loss of generality, we assume that α_i $(2 \le i \le k)$ occurs only once in each string. (Otherwise, we merge α_i with adjacent non-common regions, e.g., $\beta_{i-1}\alpha_i\beta_i$ is regarded as one non-common region). For $1 \le i \le k$, let α_i^a, α_i^b, and α_i^c be the longest suffixes of α_i occurring at least twice in the strings A, B, and C, respectively. Let α_i^* be the longest of α_i^a, α_i^b, and α_i^c. Then, these are a-suffixes of ρ, which are classified into 5 types $(1 \le i \le k)$.

1. a suffix of α_{k+1},
2. $\omega_i^a\alpha_{i+1} \dots \alpha_{k+1}$ where ω_i^a is a (non-empty) suffix of $\alpha_i^*\beta_i$.
3. $\omega_i^b\alpha_{i+1} \dots \alpha_{k+1}$ where ω_i^b is a (non-empty) suffix of $\alpha_i^*\delta_i$.
4. $\omega_i^c\alpha_{i+1} \dots \alpha_{k+1}$ where ω_i^c is a (non-empty) suffix of $\alpha_i^*\vartheta_i$.
5. $\alpha_i'(\beta_i/\,\delta_i/\,\vartheta_i)\alpha_{i+1} \dots \alpha_{k+1}$, where α_i' is a suffix of α_i longer than α_i^*.

The SAA of ρ is a lexicographically sorted list of all the a-suffixes of ρ. The SAA requires the space linear to the number of a-suffixes, i.e., $O(|A| + \sum_{i=1}^{k}(2|\alpha_i^*| + |\delta_i| + |\vartheta_i|))$ space.

For $1 \le i \le k$, let γ_i^a, γ_i^b, and γ_i^c be the longest prefix of α_{i+1} occurring at least twice in the strings A, B, and C, respectively, and let γ_i^* be the longest of γ_i^a, γ_i^b, and γ_i^c. Let B' be the concatenation of the k strings $\alpha_i^a\delta_i\gamma_i^a\#_i$ $(1 \le i \le k)$ and C' be the concatenation of the k strings $\alpha_i^a\vartheta_i\gamma_i^a\#_i$ $(1 \le i \le k)$, where $\#_i$ is a delimiter. That is,

$$B' = \alpha_1^a\delta_1\gamma_1^a\#_1\alpha_2^a\delta_2\gamma_2^a\#_2 \dots \alpha_k^a\delta_k\gamma_k^a\#_k \text{ and}$$
$$C' = \alpha_1^a\vartheta_1\gamma_1^a\#_1\alpha_2^a\vartheta_2\gamma_2^a\#_2 \dots \alpha_k^a\vartheta_k\gamma_k^a\#_k.$$

Then, Lemma 1 can be generalized to the following lemma (we omit the proof).

Lemma 2. *For every $i = 1, \dots, k$, if $|\alpha_i^b| > |\alpha_i^a|$, α_i^b occurs in B'.*

The SAA of ρ can be constructed as follows:

1. Compute $|\alpha_i^*|$ and $|\gamma_i^*|$ $(1 \le i \le k)$.
   1.1 Compute $|\alpha_i^a|$, for every $i = 1, \dots, k$, by searching for α_i^R in the suffix array of A^R and, symmetrically, compute $|\gamma_i^a|$ by searching for γ_i in the suffix array of A.
   1.2 Compute $\ell_i^b = \max(|\alpha_i^a|, |\alpha_i^b|)$ using the suffix array of $(B')^R$ as follows. Let α_i' be the longest suffix of α_i occurring in B'. We can find α_i' by searching for $(\alpha_i)^R$ in the suffix array of $(B')^R$. By Lemma 2, if $|\alpha_i'| > |\alpha_i^a|$, $\ell_i^b = |\alpha_i'|$ and, otherwise, $\ell_i^b = |\alpha_i^a|$.
   Symmetrically, compute $\max(|\gamma_i^a|, |\gamma_i^b|)$ using the suffix array of B' .
   1.3 Similarly, compute $\ell_i^c = \max(|\alpha_i^a|, |\alpha_i^c|)$ using the suffix array of $(C')^R$. Then, $|\alpha_i^*| = \max(\ell_i^b, \ell_i^c)$.
   Symmetrically, compute $\max(|\gamma_i^a|, |\gamma_i^c|)$ and $|\gamma_i^*|$.
2. Construct the GSA for $2k + 1$ strings A, $\alpha_i^*\delta_i\gamma_i^*d_i$, and $\alpha_i^*\vartheta_i\gamma_i^*d_i$, where d_i is the symbol following γ_i^* in α_{i+1} $(1 \le i \le k)$.
3. Delete suffixes of $\gamma_i^*d_i$ derived from $\alpha_i^*\delta_i\gamma_i^*d_i$ and $\alpha_i^*\vartheta_i\gamma_i^*d_i$ $(1 \le i \le k)$ in the GSA.

2.4 Pattern search

We can perform pattern search using the SAA in the same way as using classical suffix arrays of strings except for dealing with alignments in a-suffixes. Consider an a-suffix $\omega(\beta_i/\,\delta_i/\,\vartheta_i)\ldots\alpha_{k+1}$ where ω does not contain an alignment. We can perform binary search with lcp information like in classical suffix arrays until a prefix of a given pattern P matches ω. If a prefix of P matches ω, we consider only the a-suffix to search for P since ω occurs only once in each string by definition of a-suffixes. Thus, after a prefix of P matches ω, we compare P with β_i, δ_i, and ϑ_i. We can enhance this comparison using the trie of β_i, δ_i, and ϑ_i.

3 Experiments

We show by experiments that our index (the SAA) is an effective data structure for similar data. The SAA requires only about 1/7 of the space required by the GSA to index 11 human genome sequences, which is explained in the following.

3.1 Experimental data

To measure the space requirement of indexes in practice, we used one reference sequence and 10 individual sequences from 1000 Genomes project website. From the project website, we downloaded pairs of bam and bai files of 10 individual human genomes, where bam files contain *reads* (short segments of length 90-125) of each individual and bai files contain alignment of the reads. We also downloaded their corresponding reference genome, hg19. To convert a set of reads into one sequence, we used samtools[6] (Sequence Alignment/Map tools), by which we obtained 10 individual genome sequences. Since individual genome sequences are aligned to the reference genome sequence, these 11 sequences make a multiple alignment based on the reference genome sequence.

In our experiments, we used chromosome 20 of each genome. The length of the reference sequence is 63,025,520 and the lengths of the individual sequences vary from 62,965,442 to 62,965,512, which are a little shorter than the reference sequence. The sequences consist of five characters $\{A, G, T, C, N\}$, where A, G, T, and C stand for nucleotides Adenine, Guanine, Thymine, and Cytosine, respectively, and N appears in some special cases and is treated exceptionally in general (also in our experiments). In the reference sequence, N's do not appear alone but as chunks of N's. There are six chunks of N's in the reference and their lengths are 60,000, 3,100,000, 150,000, 50,000, 50,000 and 50,000. In the positions where the reference sequence has N's, individual sequences also have N's mostly. In the other positions, most of N's in individual sequences are single N's. The chunks of N's in individual sequences may represent the regions that are not sequenced, the regions that are sequenced but have very low quality, or the regions that are moved to other places. Single N's in individual sequences represent positions where one character from $\{A, G, T, C\}$ cannot be determined

[6] http://samtools.sourceforge.net/

Table 1. The number of non-common regions according to length.

Length	1	2	3	4	5	6	Total
Number	190,804	3,057	215	47	9	3	194,135

Table 2. The lengths of α^*'s and α^j's ($0 \leq j \leq 10$).

	Total length	Average length		Total length	Average length
α^0's	3,202,864	16.50	α^6's	2,987,607	15.39
α^1's	3,030,406	15.61	α^7's	3,022,359	15.57
α^2's	2,558,396	13.18	α^8's	3,132,487	16.14
α^3's	2,976,231	15.33	α^9's	3,026,544	15.59
α^4's	2,989,375	15.40	α^{10}'s	3,140,456	16.18
α^5's	2,991,517	15.14	α^*'s	3,229,589	16.64

because reads have different characters in $\{A, G, T, C\}$, there are deletions in reads, and/or the quality is low.

3.2 Experimental results

In this section, we compare the space requirements of the GSA and the SAA for the 11 sequences. For simplicity, we appended N's to the end of each individual sequence so that the length of the individual sequence is the same as that of the reference sequence. Let S_0 be the reference sequence and S_i ($1 \leq i \leq 10$) be each individual sequence. We call an aligned position a *non-common position* if at least two distinct characters in $\{A, G, T, C\}$ appear at this position. Notice that we do not regard a position as a non-common position if N and only one character in $\{A, G, T, C\}$ appear at the position. In our data set, there are 0.3% non-common positions (197,814 among 63,025,520 positions). Consecutive non-common positions become a *non-common region*. There are 194,135 non-common regions whose lengths vary from 1 to 6 (Table 1).

We first compute the lengths of α^*'s in the sequences, which is a main factor for the space requirement of our index. For a common region α_i, we denote by α_i^j the longest suffix of α_i appearing at least twice in sequence S_j ($0 \leq j \leq 10$). Recall that α_i^* is the longest of α_i^0, ..., α_i^{10}. When computing α_i^j, we exclude the part of α_i containing at least 10 consecutive N's since long consecutive N's do not carry any information about $\{A, G, T, C\}$. For each j, the total length and the average length of α_i^j's are shown in Table 2. (We omit the subscript i in α_i^j if not confusing.) For example, in sequence S_1, 3,030,406 characters (4.8%) of the entire 63,025,520 characters are included in α^1's. Since there are 194,135 non-common regions, the average length of α^1's is 15.61.

From the lengths of non-common regions and α^*'s, we calculate the space requirement of the SAA. For a substring $\alpha\beta$ of a sequence S_j where α is a common region and β is a non-common region, we call $\alpha^*\beta$ a *NS-region* (non-shared region) and α' a *S-region* (shared region) where α' is the prefix of α such that $\alpha'\alpha^*$ is α. For a sequence, let n_t be the length of the sequence, n_s be the

Table 3. Distribution of characters in our sequences.

	NS-regions (3,427,403 characters)		S-regions (59,598,117 characters)	
	# of $\{A, G, T, C\}$	# of N	# of $\{A, G, T, C\}$	# of N
S_0	3,427,401	2	56,078,119	3,519,998
S_1	3,318,768	108,635	55,263,839	4,334,278
S_2	2,940,879	486,524	49,719,051	9,879,066
S_3	3,272,652	154,751	54,872,472	4,725,645
S_4	3,279,318	148,085	54,969,949	4,628,168
S_5	3,285,414	141,989	54,972,379	4,625,738
S_6	3,275,604	151,799	54,947,026	4,651,091
S_7	3,306,405	120,998	55,010,622	4,587,495
S_8	3,385,161	42,242	55,717,092	3,881,025
S_9	3,311,346	116,057	55,045,612	4,552,505
S_{10}	3,390,329	37,074	55,722,788	3,875,329
Total	36,193,277	1,508,156	602,318,949	53,260,338

total length of S-regions, and n_n be the total length of NS-regions. (Note that n_t, n_s, and n_n are identical in all sequences and $n_t = n_s + n_n$.) Then, the size of the GSA is $11n_t$ and the size of the SAA is $n_s + 11n_n$ $(= n_t + 10n_n)$. In our data set, $n_t = 63,025,520$ and $n_n = 3,427,403$, and thus the size of the GSA is 693,280,720 words and the size of the SAA is 97,299,550 words. That is, our index uses only 14.03% space compared to the GSA.

When searching the sequences for a pattern, we may assume that the pattern does not contain N since we do not consider wild-card matches. In this circumstance, we can reduce the space requirement of indexes by eliminating in indexes the suffixes whose first characters are N. To compute the sizes of the two indexes for our data set, we first investigate the distribution of N in our sequences. Table 3 shows the distribution of characters in NS-regions and S-regions for each sequence. For example, in NS-regions of sequence S_1, the number of characters A, G, T, C is 3,318,768 (97%) and the number of character N is 108,635 (3%). In S-regions of sequence S_1, the number of characters A, G, T, C is 55,263,839 (93%) and the number of character N is 4,334,278 (7%).

We compute the sizes of the two indexes when excluding the suffixes whose first characters are N. The size of the GSA is the total number of characters A, G, T, C in NS-regions and S-regions of the 11 sequences, which is 638,512,226 $(36,193,277 + 602,318,949)$ words (see Table 3). Next, consider the SAA. For a position in an NS-region, we eliminate the suffix of each sequence starting at this position if the first character of the suffix is N. For a position in an S-region, we eliminate the suffix (a-suffix) starting at this position only if the characters in the position are N in all sequences. In our data set, the total number of A, G, T, C in NS-regions is 36,193,277 and the number of positions in S-regions excluding the positions where characters are N in all sequences is 56,078,133 (see the last row in Table 4). Thus, the size of our index is 92,271,410 words, which is only 14.45% of the size of the GSA.

Table 4. Comparison of the sizes of the GSA and the SAA according to the number of sequences when excluding the suffixes whose first characters are N. Column (C1) is the total number of A, G, T, C in NS-regions and column (C2) is the number of positions in S-regions excluding the positions where characters are N in all sequences. Then, the size of the SAA is (C1) + (C2). The ratio of the size of the SAA to that of the GSA is given in the last column.

	Size of GSA	Size of SAA	(C1)	(C2)	Ratio (%)
$S_0 \sim S_1$	118,088,127	60,455,692	1,914,833	58,540,859	51.20
$S_0 \sim S_2$	170,748,057	62,473,223	4,553,672	57,919,551	36.59
$S_0 \sim S_3$	228,893,181	65,338,136	7,905,004	57,433,132	28.55
$S_0 \sim S_4$	287,142,448	68,758,063	11,706,204	57,051,859	23.95
$S_0 \sim S_5$	345,400,241	72,483,002	15,719,471	56,763,531	20.99
$S_0 \sim S_6$	403,622,871	76,417,395	19,881,933	56,535,462	18.93
$S_0 \sim S_7$	461,939,898	80,305,819	23,928,531	56,377,288	17.38
$S_0 \sim S_8$	521,042,151	84,226,410	27,963,745	56,262,665	16.16
$S_0 \sim S_9$	579,399,109	88,223,461	32,062,878	56,160,583	15.23
$S_0 \sim S_{10}$	638,512,226	92,271,410	36,193,277	56,078,133	14.45

We also compare the space requirements of the GSA and the SAA according to the number of sequences used in indexing (Table 4). Obviously, the space efficiency of our index increases as the number of the sequences increases. The ratio of the space of the SAA to that of the GSA is 51.2% when two sequence are used, and the ratio is 14.45% when the 11 sequences are used.

Acknowledgements

Joong Chae Na was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology(2012-0003214), and by the IT R&D program of MKE/KEIT [10038768, The Development of Supercomputing System for the Genome Analysis]. Heejin Park was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology(2012-0006999), by Seoul Creative Human Development Program (HM120006), by the Proteogenomics Research Program through the National Research Foundation of Korea funded by the Korean Ministry of Education, Science and Technology, and by the National Research Foundation of Korea(NRF) funded by the Ministry of Science, ICT & Future Planning(2012-054452). Laurent Mouchard was supported by the French Ministry of Foreign Affairs Grant 27828RG (INDIGEN, PHC STAR 2012). Kunsoo Park was supported by National Research Foundation of Korea- Grant funded by the Korean Government(MSIP) (2012K1A3A4A07030483), and by Next-Generation Information Computing Development Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Science, ICT & Future Planning (2011-0029924).

References

1. M. I. Abouelhoda, S. Kurtz, and E. Ohlebusch. Replacing suffix trees with enhanced suffix arrays. *Journal of Discrete Algorithms*, 2(1):53–86, 2004.
2. A. Apostolico. The myriad virtues of subword trees. In A. Apostolico and Z. Galil, editors, *Combinatorial Algorithms on Words*, pages 85–95. Springer, 1985.
3. M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, Paolo Alto, California, 1994.
4. The 1000 Genomes Project Consortium. A map of human genome variation from population-scale sequencing. *Nature*, 467(7319):1061–1073, 2010.
5. H. H. Do, J. Jansson, K. Sadakane, and W. K. Sung. Fast relative Lempel-Ziv self-index for similar sequences. In *Proceedings of FAW-AAIM 2012*, pages 291–302, 2012.
6. P. Ferragina and G. Manzini. Indexing compressed text. *Journal of the ACM*, 52(4):552–581, 2005.
7. R. Grossi and J. S. Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. *SIAM Journal on Computing*, 35(2):378–407, 2005.
8. D. Gusfield. *Algorithms on Strings, Tree, and Sequences*. Cambridge University Press, Cambridge, 1997.
9. S. Huang, T. W. Lam, W. K. Sung, S. L. Tam, and S. M. Yiu. Indexing similar dna sequences. In *Proceedings of AAIM 2010*, pages 180–190, 2010.
10. J. Kärkkäinen, P. Sanders, and S. Burkhardt. Linear work suffix array construction. *Journal of the ACM*, 53(6):918–936, 2006.
11. D. K. Kim, M. Kim, and H. Park. Linearized suffix tree: an efficient index data structure with the capabilities of suffix trees and suffix arrays. *Algorithmica*, 52(3):350–377, 2008.
12. D. K. Kim, J. S. Sim, H. Park, and K. Park. Constructing suffix arrays in linear time. *Journal of Discrete Algorithms*, 3(2–4):126–142, 2005.
13. P. Ko and S. Aluru. Space efficient linear time construction of suffix arrays. *Journal of Discrete Algorithms*, 3(2–4):143–156, 2005.
14. S. Kreft and G. Navarro. On compressing and indexing repetitive sequences. *Theoretical Computer Science*, 483:115-133, 2013.
15. S. Kuruppu, S. J. Puglisi, and J. Zobel. Relative Lempel-Ziv compression of genomes for large-scale storage and retrieval. In *Proceedings of SPIRE 2010*, pages 201–206, 2010.
16. V. Mäkinen, G. Navarro, J. Sirén, and N. Välimäki. Storage and retrieval of individual genomes. In *Proceedings of RECOMB 2009*, pages 121–137, 2009.
17. V. Mäkinen, G. Navarro, J. Sirén, and N. Välimäki. Storage and retrieval of highly repetitive sequence collections. *Journal of Computational Biology*, 17(3):281–308, 2010.
18. U. Manber and G. Myers. Suffix arrays: A new method for on-line string searches. *SIAM Journal on Computing*, 22(5):935–948, 1993.
19. J. C. Na, M. Crochemore H. Park, J. Holub, C. S. Iliopoulos, L. Mouchard, and K. Park. Suffix tree of alignment: An efficient index for similar data. In *Proceedings of IWOCA 2013*, 2013.
20. G. Navarro. Indexing highly repetitive collections. In *Proceedings of IWOCA 2012*, pages 274–279, 2012.
21. J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.