Special issue: Theoretical computer science and discrete mathematics

Research Paper

Entropy estimation with suffix arrays

Kunihiko SADAKANE[1]
[1]*National Institute of Informatics*

ABSTRACT

In this paper we give an algorithm for estimating the entropy of a string using the suffix array. Our algorithm is based on a new combinatorial property of the longest common prefix array of a string.

KEYWORDS

Strings, entropy estimation, suffix arrays

1 Introduction

This paper proposes a fast algorithm for entropy estimation of a string. There exists an entropy estimation scheme based on the LZ77 compression algorithm (See Section 2.2 for details). However, the LZ77 compression algorithm requires dynamic data structures which are in practice slow. The algorithm proposed in this paper uses a static data structure called a suffix array which can be constructed quickly, and returns the same estimation value with that using LZ77. Our data structure can also compute another estimation value of the entropy (See Section 2.1).

2 Known algorithms for entropy estimation

In this section we review two known algorithms for entropy estimation for strings. Throughout this paper, we use the following definitions and notations. For a string $S = S[0]S[1]\ldots$, let S_j denote $S[j]S[j+1]\ldots$ and it is called a suffix of S. Let $lcp(\alpha,\beta)$ denote the length of the longest common prefix of two strings α and β.

2.1 Using distinguishing prefixes

Definition 1 *For a string S, we define distinguishing prefix of a suffix S_j as the shortest prefix of S_j that does not match with prefixes of any other suffixes. We also define B_j as the length of the distinguishing prefix of the suffix S_j.*

Example: Let $S[0..13]$ = "tobeornottobe$". The distinguishing prefix of a suffix S_8 = "ttobe$" is "tt" and therefore $B_8 = 2$.

Yokoo [1], [2] proposed an algorithm to incrementally estimate the entropy of a string. He uses the average value of B_j to estimate the entropy. He also proposed an expected linear time algorithm to maintain a list of prefixes which are in lexicographic order of reversed strings. Though it seems to be natural to estimate the entropy by B_j, Yokoo showed that the entropy estimated by the average value of B_j is sometimes far from real compression ratio of many compression algorithms. Therefore we use another entropy estimator.

2.2 Using the LZ77 algorithm

Definition 2 *We define $L_n(j)$ as the smallest integer $L(>0)$ which satisfies $x_j^{j+L-1} \neq x_i^{i+L-1}$, $j-n \leq \forall i < j$.*

$L_n(j)$ stands for the length of a prefix of a suffix S_j such that the prefix does not match with prefixes of suffixes S_{j-n} to S_{j-1}. For a random string X, let $L_n(X)$ denote a random variable for $L_n(0)$.

Example: Let $S[0..13]$ = "tobeornottobe$". Since S_9 = "to be$" matches with the prefix of S_0 by 4 characters, $L_9(9) = 5$. On the other hand, S_9 matches with a prefix of suffixes $S_1, \ldots, S_8$ by 1 character, and therefore $L_8(9) = 2$.

The difference between B_j and $L_n(j)$ is that $L_n(j)$ is defined as the length of prefix of S_j which does not appear in prefixes of suffixes preceding S_j, while B_j is defined as distinguishing length between S_j and both

Received November 1, 2011; Accepted December 22, 2011.
[1)] *sada@nii.ac.jp, http://researchmap.jp/sada/*
DOI: 10.2201/NiiPi.2012.9.4

preceding and following suffixes. The value $L_n(j)$ corresponds to the length of a prefix of S_j encoded by the LZ77 algorithm [3]. The following theorem proves asymptotical optimality of the LZ77 code.

Theorem 1 (Wyner, Ziv [4]) *As* $n \to \infty$,

$$\frac{\log n}{L_n(X)} \to H, \text{ in probability,}$$

where H is the entropy of a stationary ergodic information source.

We use the average value of $L_j(j)$

$$\sum_{j=0}^{n-1} L_j(j)$$

to estimate the entropy. Note that we use not $L_n(j)$ but $L_j(j)$ because we consider a finite string $x_0, \ldots, x_n$.

3 A Combinatorial property of longest common prefix array

In this section we give a new combinatorial property of the longest common prefix array. First we give definitions. Let $S = S[0]S[1]\ldots S[n]$ be a string. The suffix array [5] of S is an integer array $I[0..n]$ such that $I[i] = j$ if $S_j = S[j]S[j+1]\ldots S[n]$ is lexicographically i-th suffix among all suffixes of S. The longest common prefix array is an array storing $lcp(I[i-1], I[i]) + 1$ for $i = 0, \ldots, n$.

3.1 Main result

The following is the main theorem of this paper, which connects a combinatorial value defined on the suffix array with an information theoretic value appeared in the LZ77 compression algorithm.

Theorem 2

$$\sum_{i=1}^{n}\{lcp(I[i-1], I[i]) + 1\} = \sum_{j=0}^{n-1} L_j(j)$$

To prove this theorem, we first show that the summation of lcp's in the lemma does not change after rearrangement of suffixes in the suffix array according to a particular order, then show that the equation holds in the rearranged array I'.

We define an order $<_{LZ}$ for the rearrangement. The order of suffixes of a string for the rearrangement is defined as follows.

Definition 3 *A suffix* S_s *is smaller than a suffix* S_t *in order of* $<_{LZ}$ *if the following condition holds. Let* $l = lcp(S_s, S_t)$ *and*

$$j_s = \operatorname*{argmin}_{0\le j\le n-1}\{j | lcp(S_j, S_s) \ge l + 1\}$$

$$j_t = \operatorname*{argmin}_{0\le j\le n-1}\{j | lcp(S_j, S_t) \ge l + 1\},$$

then

$$j_s < j_t.$$

Note that the string has a unique terminator \$ and therefore any suffix is not contained in other suffixes. Thus $l < \min\{|S_s|, |S_t|\}$ and the sets in the definition of j_s and j_t have at least one element, s and t, respectively. Therefore j_s and j_t are well-defined.

To sort suffixes according to the order $<_{LZ}$, we first divide the suffixes according to their first characters into several groups and arrange the groups. The order of the groups is defined by minimum values of indices of the suffixes in the groups. These values correspond to j_s and j_t in the definition of $<_{LZ}$. We assume that suffixes S_a and S_b are in the same group G_1 and suffixes S_a and S_c are in different groups G_1 and G_2. Then $lcp(S_a, S_b) \ge 1$ and $lcp(S_a, S_c) = 0$. Indices j_a and j_c in the definition of the order of S_a and S_c are equal for all suffixes S_a in G_1 and S_c in G_2. The indices j_a and j_c become the smallest indices in G_1 and G_2, respectively. This justifies the definition of order of groups.

Next we subdivide the suffixes in the groups according to their second characters. The order of the groups is also defined by minimum indices in the groups. We continue dividing suffixes until all suffixes belong to different groups.

Fig. 1 and Fig. 2 show an example of I and I' for the string 'to be or not to be \$.' The longest-match suffix of a suffix $S_s = S_4$ is $S_t = S_1$. In Fig. 2, $t = I'[3]$ and $s = I'[5]$ and $L_s(s) = 2$ appears in $lcp(I'[5-1], I'[5]) + 1$. We see that in Fig. 2, $lcp(I[i-1], I[i]) + 1 = L_s(s)$ $(s = I'[i])$ for all i.

Proposition 1 *The order* $<_{LZ}$ *is transitive, that is, if* $S_a <_{LZ} S_b$ *and* $S_b <_{LZ} S_c$, *then* $S_a <_{LZ} S_c$ *for all suffixes* S_a, S_b, *and* S_c.

Proof: We assume that

$$l_1 = lcp(S_a, S_b)$$
$$j_a = \operatorname*{argmin}_{0\le j\le n-1}\{j | lcp(S_j, S_a) \ge l_1 + 1\}$$
$$j_{b1} = \operatorname*{argmin}_{0\le j\le n-1}\{j | lcp(S_j, S_b) \ge l_1 + 1\}$$
$$l_2 = lcp(S_b, S_c)$$
$$j_{b2} = \operatorname*{argmin}_{0\le j\le n-1}\{j | lcp(S_j, S_b) \ge l_2 + 1\}$$
$$j_c = \operatorname*{argmin}_{0\le j\le n-1}\{j | lcp(S_j, S_c) \ge l_2 + 1\}$$
$$l_3 = lcp(S_a, S_c)$$
$$j'_a = \operatorname*{argmin}_{0\le j\le n-1}\{j | lcp(S_j, S_a) \ge l_3 + 1\}$$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
$s = I[i]$	13	11	2	12	3	6	10	1	4	7	5	9	0	8
	$	b	b	e	e	n	o	o	o	o	r	t	t	t
		e	e	$	o		b	b	r	t		o	o	t
		$	o		r		e	e				b	b	
							$	o				e	e	
												$	o	
$lcp(I[i-1], I[i]) + 1$	1	1	3	1	2	1	1	4	2	2	1	1	5	2
$L_s(s)$	1	3	1	2	1	1	4	1	2	2	1	5	1	2
B_s	1	3	3	2	2	1	4	4	2	2	1	5	5	2

Fig. 1 Alphabetic order.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
$s = I'[i]$	0	9	8	1	10	4	7	2	11	3	12	5	6	13
	t	t	t	o	o	o	o	b	b	e	e	r	n	$
	o	o	t	b	b	r	t	e	e	o	$			
	b	b		e	e			o	$					
	e	e		o	$									
	o	$												
$lcp(I'[i-1], I'[i]) + 1$	1	5	2	1	4	2	2	1	3	1	2	1	1	1
$L_s(s)$	1	5	2	1	4	2	2	1	3	1	2	1	1	1
B_s	5	5	2	4	4	2	2	3	3	2	2	1	1	1

Fig. 2 $<_{LZ}$ order.

$$j'_c = \underset{0 \le j \le n-1}{\operatorname{argmin}} \{ j | lcp(S_j, S_c) \ge l_3 + 1 \}$$
$$j_a < j_{b1}$$
$$j_{b2} < j_c.$$

We show that $j'_a < j'_c$. There are three cases.

Case $l_1 < l_2$: $lcp(S_b, S_c) = l_2 \le l_1 + 1$ and $l_3 = l_1$. Therefore $j_a = j'_a$, and j_{b1} and j'_c become as follows:

$$j_{b1} = \underset{j}{\operatorname{argmin}} \{ j | lcp(S_j, S_b) \ge l_1 + 1, 0 \le j \le n-1 \}$$
$$j'_c = \underset{j}{\operatorname{argmin}} \{ j | lcp(S_j, S_c) \ge l_1 + 1, 0 \le j \le n-1 \}.$$

Because $lcp(S_b, S_c) \le l_1 + 1$, definitions of j_{b1} and j'_c coincide and therefore $j_{b1} = j'_c$. These show that $j'_a = j_a < j_{b1} = j'_c$.

Case $l_1 > l_2$: we can show that $j'_a = j_{b2} < j_c = j'_c$ in a similar way.

Case $l_1 = l_2$: From the definition $j_{b1} = j_{b2}$ and $l3 \ge l1$. If $l_3 = l_1 = l_2$, $j'_a = j_a < j_{b1} = j_{b2} < j_c = j'_c$. If $l_3 > l_1 = l_2$, $lcp(S_a, S_c) \ge l_1 + 1$ and definitions of j_a and j_c become as follows:

$$j_a = \underset{j}{\operatorname{argmin}} \{ j | lcp(S_j, S_a) \ge l_1 + 1, 0 \le j \le n-1 \}$$
$$j_c = \underset{j}{\operatorname{argmin}} \{ j | lcp(S_j, S_c) \ge l_1 + 1, 0 \le j \le n-1 \}.$$

These definitions coincide and therefore $j_a = j_c$. Then $j_{b2} < j_c = j_a < j_{b1}$, which contradicts $j_{b1} = j_{b2}$. It means that the case $l_3 > l_1 = l_2$ does not occur.

In all cases, the inequality $j'_a < j'_c$ holds. □

After this rearrangement, the summation of lcp's between adjacent suffixes in I' is equal to that in the original suffix array I.

Lemma 1

$$\sum_{i=1}^{n} lcp(I[i-1], I[i]) = \sum_{i=1}^{n} lcp(I'[i-1], I'[i])$$

Proof: We show that lcp's of adjacent suffixes in I' are a permutation of that in I. After dividing suffixes into groups according to the first characters, lcp's between different groups are zero and they do not depend on sorting order. Therefore the number of zeroes in lcp's does not depend on sorting order. Then suffixes in each group are subdivided according to the second characters. After the subdivision lcp between two subgroups in a group is one regardless of sorting order. Therefore the number of ones in lcp's does not change. We can prove that the number of l in lcp's between adjacent suffixes in I' is always equal to that in I in a similar way. □

Now we can show a relationship between lcp and $L_s(s)$. Note that we use $L_s(s)$ instead of $L_n(s)$.

Lemma 2 *Concerning a suffix S_s ($s = I'[i]$), a suffix S_t which corresponds to $L_s(s)$ is located in smaller position than S_s in the array I', and*

$$lcp(I'[i-1], I'[i]) + 1 = L_s(s).$$

Note that the suffix S_t may not be adjacent to S_s.

Proof: From the definition of the order $<_{LZ}$, one of suffixes which are the longest-match suffixes of S_s is located in $I'[i-1]$ or $I'[i+1]$. Suffixes which are larger than S_s in order of $<_{LZ}$, that is, suffixes $I'[i+k](k=1,2,\ldots)$ do not correspond to $L_s(s)$ even if $lcp(I'[i], I'[i+1]) > lcp(I'[i], I'[i-1])$. The reason is as follows. We assume that a suffix S_u is in $I'[i+k](k=1,2,\ldots)$. Let $lcp(S_s, S_u) = l$ and

$$j_s = \underset{0\le j\le n-1}{\operatorname{argmin}}\{j | lcp(S_j, S_s) \ge l_1 + 1\}$$
$$j_u = \underset{0\le j\le n-1}{\operatorname{argmin}}\{j | lcp(S_j, S_u) \ge l_1 + 1\}.$$

Since $S_s <_{LZ} S_u$, $j_s < j_u$. It means that the suffix S_u is not a suffix which corresponds to $L_s(s)$.

Therefore suffixes $I'[i-k](k=1,2,\ldots)$ are candidates for the longest-match suffix of S_s. Some of them are excluded because their positions are larger than s. However, there exists at least one suffix S_t for each suffix S_s except S_0 such that $t < s$ and $lcp(S_t, S_s) = lcp(I'[i-1], I'[i])$. Since $L_s(s) = lcp(S_t, I'[i]) + 1$, $lcp(I'[i-1], I'[i]) + 1 = L_s(s)$. □

The proof of Theorem 2 is obvious from Lemma 1 and Lemma 2.

3.2 Linear time algorithm

The value $\sum_{j=0}^{n-1} L_j(j)$ is computed in linear time as follows. First we construct the suffix array I in linear time using a practical linear time algorithm [6]. Then the lcp array can be also computed in linear time [7]. It is obvious to obtain the desired value from the lcp array in linear time.

3.3 Relation with distinguishing prefixes

We also show the relation between the summation of B_j and the summation of $L_j(j)$. B_j can be computed by

$$B_j = \max\{lcp(I[i-1], I[i]) + 1, \; lcp(I[i], I[i+1]) + 1\}.$$

Therefore the entropy estimation with this scheme is also done in linear time.

We express an upper-bound of the summation of B_j's for all suffixes by the summation of lcp's between adjacent suffixes in the suffix array.

Lemma 3

$$\sum_{j=0}^{n-1}(B_j - 1) \le 2\sum_{i=1}^{n} lcp(I[i-1], I[i])$$

Proof: By using an inequality

$$\max\{a, b\} \le a + b \quad (a, b \ge 0),$$

i	0	1	2	3	4	5	6
$j = I[i]$	6	3	0	4	1	5	2
	\$	a	a	b	b	c	c
		b	b	c	c	\$	a
		c	c	\$	a		b
		\$	a		b		c
			b		c		\$
			c		\$		
			\$				
$lcp(I[i-1], I[i])$	0	0	3	0	2	0	1
$B_j - 1$	0	3	3	2	2	1	1

Fig. 3 An example of a string.

$$\begin{aligned}
&\sum_{j=0}^{n-1}(B_j - 1) \\
&= \sum_{i=1}^{n}\max\{lcp(I[i-1], I[i]), lcp(I[i], I[i+1])\} \\
&\le \sum_{i=1}^{n}\{lcp(I[i-1], I[i]) + lcp(I[i], I[i+1])\} \\
&= 2\sum_{i=1}^{n} lcp(I[i-1], I[i])
\end{aligned}$$

Note that $lcp(I[0], I[1]) = lcp(I[n], I[n+1]) = 0$. □

The equality holds when the string X forms concatenation of the same two strings in which all characters differ, for example 'abc...zabc...z\$.' Fig. 3 shows an example of a suffix array for such a string. for a string of length $2n$. In such cases,

$$\sum_{i=1}^{2n} lcp(I[i-1], I[i]) = \frac{n(n+1)}{2}$$
$$\sum_{j=0}^{2n-1} B_j = n(n+1)$$

This means that estimated values of entropy by B_j is sometimes far from that by $L_j(j)$.

References

[1] H. Yokoo, "Context tables: a tool for describing text compression algorithms," In *Proc. of IEEE Data Compression Conference*, pp.299–308, 1998.

[2] H. Yokoo, "A Dynamic data structure for reverse lexicographically sorted prefixes," In M. Crochemore and M. Paterson, editors, *Proc. of the 10th Annual Symposium on Combinatorial Pattern Matching (CPM'99)*, LNCS 1645, pp.150–162, 1999.

[3] J. Ziv and A. Lempel, "A Universal algorithm for sequential data compression," *IEEE Trans. Inform. Theory*, vol.IT-23, no.3, pp.337–343, 1977.

[4] A. D. Wyner and J. Ziv, “Some Asymptotic Properties of the Entropy of a Stationary Ergodic Data Source with Applications to Data Compression,” *IEEE Trans. Inform. Theory*, vol.IT-35, no.6, pp.1250–1258, 1989.

[5] U. Manber and G. Myers, “Suffix arrays: A new method for on-line string searches,” *SIAM Journal on Computing*, vol.22, no.5, pp.935–948, 1993.

[6] G. Nong, S. Zhang, and W. H. Chan, “Linear suffix array construction by almost pure induced-sorting,” In *DCC*, pp.193–202, 2009.

[7] S. Gog and E. Ohlebusch, “Fast and lightweight lcp-array construction algorithms,” In *Proc. ALENEX*, pp.25–34, 2011.

Kunihiko SADAKANE

Kunihiko SADAKANE received B.S., M.S., and Ph.D. degrees from Department of Information Science, University of Tokyo in 1995, 1997 and 2000, respectively. He was a research associate at Graduate School of Information Sciences, Tohoku University from 2000 to 2003, and an associate professor at Faculty of Information Science and Electrical Engineering, Kyushu University from 2003 to 2009. Since 2009, he has been an associate professor at National Institute of Informatics. His research interest includes information retrieval, data structures, and data compression. He is a member of IPSJ and IEICE.