Full-Text Search Using Double-Array CDAWG

Yuma Fujita, Yoshiaki Ichihashi, Shunsuke Kanda, Kazuhiro Morita, and Masao Fuketa

***Abstract*—Double-Array is a method widely used for handling sets of strings. While the method can conduct fast retrieval, there are not application examples for Full-Text Search. Compact directed acyclic word graph (CDAWG) is a data structure preserving some features of directed acyclic word graph (DAWG), and requires less space than DAWG. When using CDAWG for Full-Text Search, it can make a graph to conduct fast retrieval not depending on a text size. A method that represents DAWG using Double-Array has been proposed. Therefore, we propose a new method using Double-Array CDAWG for high speed Full-Text Search. Experimental results show the effectiveness of the proposed method.**

***Index Terms*—CDAWG, double-array, full-text search, genome search.**

I. Introduction

Techniques that search need data are required for extensively increasing data. Full-Text Search is one of the techniques. Full-Text Search is a technique for searching substrings from a full-text database. For example, Full-Text Search is used for web searches and genome searches. While compressed suffix array (CSA) [1]-[3] excels in existing methods [4], the retrieval time depends on a text size. Therefore, CSA is not suitable for large scale documents in retrieval time. By contrast, a suffix compact directed acyclic word graph (CDAWG) implements Full-Text Search using a graph which handles all suffixes of a text. CDAWG [5], [6] is a data structure for a more compact representation of directed acyclic word graph (DAWG) [7]. The suffix CDAWG has an advantage that can conduct fast retrieval not depending on a text size in Full-Text Search. However, a size of the graph is larger than an array size of CSA.

In this paper, we propose high speed Full-Text Search using Double-Array [8]. A trie [9] is a data structure for handling sets of strings. Double-Array can conduct fast retrieval in the trie. Additionally, Double-Array can also represent DAWG [10]. However, there is not a method that represents CDAWG using Double-Array. Hence, we propose Double-Array representing suffix CDAWG and applies it for high speed Full-Text Search.

Manuscript received November 14, 2016; revised December 30, 2016.

Yuma Fujita, Shunsuke Kanda, Kazuhiro Morita, and Masao Fuketa are with the Department of Information Science and Intelligent Systems, Faculty of Engineering, Tokushima University, 2-1 Minamijosanjima-cho, Tokushima-shi, Tokushima 770-8506, Japan.

Yoshiaki Ichihashi was with the Department of Information Science and Intelligent Systems, Faculty of Engineering, Tokushima University. He is now with Toshiba Solutions Corporation, 72-34 Horikawa-cho, Saiwai-ku, Kawasaki-shi, Kanagawa, 212-8585, Japan (e-mail: yuma@jo-studio.com).

II. Full-Text Search and Double-Array

A. Outline of Full-Text Search

In text retrieval, Full-Text Search refers to techniques for searching substrings from a full-text database. When dealing with a large number of documents, Full-Text Index that is a popular technique, divides the documents into two tasks: indexing and searching. The indexing scans texts all of the documents and builds a list of search terms. In the searching, the index is referenced to obtain a specific query. In Full-Text Index, there are methods using all suffixes of the full-text to search a query fully and fast. For example, the methods include a suffix trie, a suffix tree, a suffix DAWG, and a suffix CDAWG.

A suffix trie, a suffix tree, a suffix DAWG, and a suffix CDAWG for a finite string w are denoted as *Trie*(w), *Tree*(w), *DAWG*(w), and *CDAWG*(w), respectively. *Trie*(w) is a data structure to represent every suffix of w in a trie. The trie is an ordered tree data structure used to store strings. There are two operations, *compaction* and *minimization*, to reduce trie nodes. Operation *compaction* compacts longest paths consisting of nodes having one child. Operation *minimization* identifies isomorphic subtrees. Operations *compaction* and *minimization* to *Trie*(w) lead to *Tree*(w) and *DAWG*(w), respectively. *CDAWG*(w) is built from either *compaction* to *DAWG*(w) or *minimization* to *Tree*(w).

Fig. 1 shows *Tree*(w) , *DAWG*(w) , and *CDAWG*(w) by setting w = "gtagtaaac". A special end marker symbol '$' is used at the end of strings in the examples. Figs.1 (a) and (b) show that there is not any difference between the amounts of nodes in the *Tree*(w) and the *DAWG*(w). It is hard to choose which operation is an effective technique for a finite string. Fig. 1 (c) shows the number of nodes in the *CDAWG*(w) is the smallest of the 3 methods because the *CDAWG*(w) is given by the trie(w) with *compaction* and *minimization*. Therefore, *CDAWG*(w) is an efficient data structure in the Full-Text Search.

B. Outline of Double-Array

Double-Array uses two one-dimensional arrays, named BASE and CHECK, to represent trie nodes. Elements BASE[s] and CHECK[s] correspond to node s. The following equations represent an arc from node s to node t with character c.

$$t = \mathrm{BASE}[s] + c \text{ and } \mathrm{CHECK}[t] = c. \quad (1)$$

The position of the destination node t is calculated by the sum of the offset BASE[s] and the numerical code of character c. The character c is stored in CHECK[t]. In

doi: 10.18178/ijfcc.2016.5.6.478

different nodes *s* and *s'*, if BASE[*s*] and BASE[*s'*] are the same, these are traversed to the same node *t*. Accordingly, all BASE values need to be constructed as different values. Equation (1) does not allow of a transition with a string or over 2 characters.

- Meanwhile, a DAWG is a data structure to identify isomorphic subtrees in a trie. Since, there is a possibility that some nodes have the same destination node in a DAWG, Double-Array cannot represent an original DAWG. Yata *et al*. presented a method that is a designed DAWG for Double-Array [10].

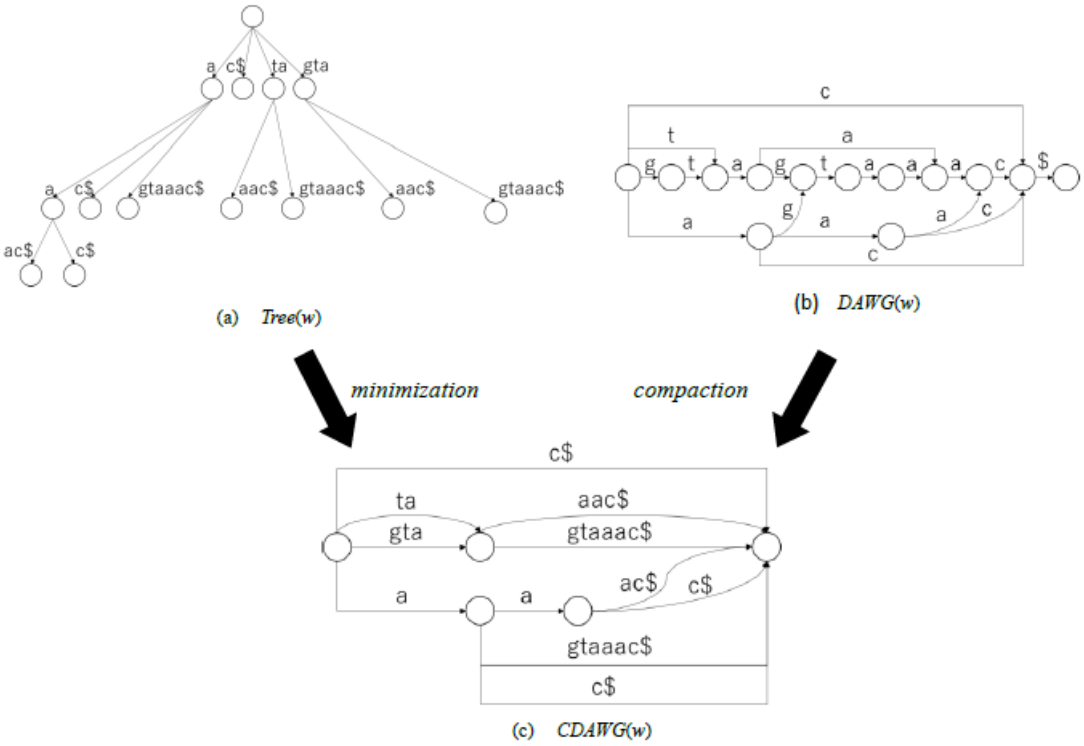

Fig. 1. Three methods for *w*.

Fig. 2 shows the designed *DAWG*(*w*). Values of {'$', 'a', 'c', 'g', 't'} are denoted {0, 1, 2, 3, 4} in Equation (1), respectively. To use Double-Array, there are a few more the nodes than the original *DAWG*(*w*) of Fig. 1 (b). In the traversal from nodes 4 and 15 to node 16, Equation (1) is satisfied because their arc labels are the same.

III. Proposed Method

As described in Section II, *CDAWG*(*w*) is an efficient data structure in Full-Text Search. We propose a new method that is a suffix CDAWG using Double-Array for high speed Full-Text Search. In some cases, an arc label is string in a CDAWG. It is necessary to represent the string labels for applying Double-Array to CDAWG representation. Fig. 1 (c) shows that there are two types of the arc strings including the special end marker symbol '$' or not, named *terminal string* and *internal string*, respectively. The proposed method has representations for them. Sections 3.1 and 3.2 show how to represent terminal strings and internal strings, respectively.

A. TAIL

General Double-Array implementation stores each suffix, which corresponds to only a registered string in a trie, to an array TAIL in order to reduce the number of nodes. The common suffixes are merged. The terminal strings become common suffixes in a text. Therefore, TAIL is suitable for storing terminal strings. Fig. 3 shows *CDAWG*(*w*) in Fig.1 (c) is designed for Double-Array. The CDAWG requires less nodes than the designed DAWG of Fig. 2. Additionally, Fig. 3. shows TAIL represents terminal strings in the *CDAWG*(*w*). The seven terminal strings are integrated into the TAIL in Fig. 3.

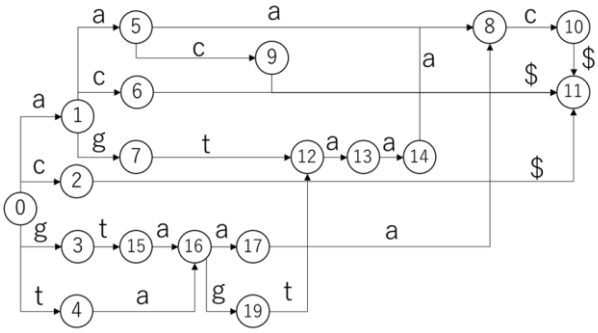

Fig. 2. Designed *DAWG*(*w*).

B. Representation of Internal Strings

This subsection discusses representation of internal strings. The proposed method divides an internal string into the first character and the rest. The first character and the rest of a string label to node *s* are stored in CHECK[*s*] and a new array called REMAIN, respectively. Furthermore, we prepare two bit arrays, named *B* and *S*, for referring to REMAIN from CHECK. *B*[*s*] is set to 1 when the arc label to node *s* is string. *S*[*i*] is set to 1 when REMAIN[*i*] corresponds to the initial position of the rest. The following operations [11] are used to

refer to REMAIN from CHECK.

- *P*.*rank*(*i*) returns the number of occurrences of 1 in *P*[0, *i*).
- *P*.*select*(*i*) returns the position of the *i* + 1-th occurrence of 1 in *P*.
- *S*.*select*(*B*.*rank*(*s*)) calculates the initiation position from node *s*.

Fig. 4 shows CHECK and REMAIN for *CDAWG*(*w*) in Fig. 3. Consider the traversal from node 0 to node 3 with "gta" in Fig. 3. The first character and the rest are 'g' and "ta", respectively. First, the first character is given by CHECK[3] = 'g'. There is the rest by *B*[3] = 1. The initiation position from node 3 is 0 by *S*.*select*(*B*.*rank*(3)). Thus, the first character of the rest '*t*' is obtained by REMAIN[0]. *S*[1] = 0 indicates that it is necessary to search REMAIN[1]. Similarly, the second character is obtained by REMAIN[1] = 'a'. From *S*[2] = 1, the internal string from node 3 is "gta".

IV. Evaluation

A. Experimental Details

We verified the effectiveness of the proposed method. As described in Section I, CSA excels in existing methods [4].

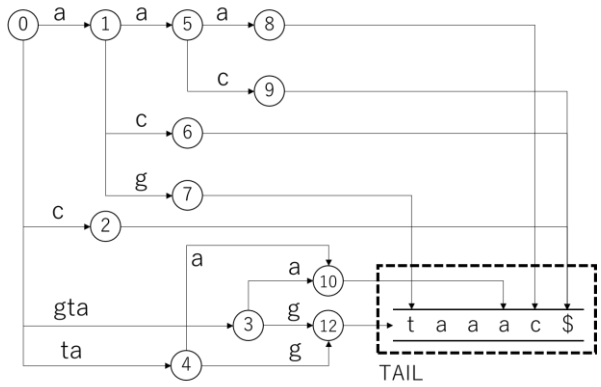

Fig. 3. *CDAWG*(*w*) in Fig. 1 is designed for Double-Array with TAIL.

	0	1	2	3	4	5	…
CHECK		a	c	g	t	a	…
B:	0	0	0	1	1	0	…

	0	1	2
REMAIN	t	a	a
S:	1	0	1

Fig. 4. CHECK and REMAIN for *CDAWG*(*w*) in Fig. 2.

For this reason, we used CSA provided from csalib100810 [12] as a comparative method. CSA and the proposed method were implemented with C and C++, respectively. Experiments for retrieval speed and index size were executed on Windows10 Pro based on Intel (R) Core (TM) i7-4785T CPU@2.20GHz (RAM : 16GB). The experiments used 50MB of the DNA data and 50MB of the ENGLISH data from Pizza&Chili Corpus [13]. We extracted 100,000 characters from the beginning of them as full text documents. We prepared 100,000 queries extracted from the random positions of the documents each. The queries have four pattern lengths of 10, 30, 60, and 90 characters. The retrieval times were measured for the 100,000 queries. Each test was averaged on 10 runs. The options of CSA were set to "-P3:512 –I:128:256" for the DNA data and "-P4:512 –I:128:256" for the ENGLISH data.

B. Results and Discussion

Table I shows the retrieval time for each document. When the query-length is 10 characters in the ENGLISH data, the proposed method can conduct the retrieval 201 times faster than CSA. When query-length is 90 characters, the proposed method can conduct the retrieval 1400 times faster than CSA. As can be seen from the result, the length of a query doesn't affect the retrieval time in the proposed method.

Table II shows the index size for each document. The index sizes of the proposed method are 21 and 9 times larger than those of CSA in the DNA data and ENGLISH data, respectively. However, the proposed method excels in trade-off between the retrieval time and the index size. When query-length is 10 characters in the retrieval time for the DNA data, the proposed method can conduct the retrieval 124 times faster than CSA even though the result is the smallest difference in Table I. From these results, the proposed method is efficient for high speed Full-Text-Search.

TABLE I: Retrieval Time for Each Document

Retrieval time (s)		query-length			
		10	30	60	90
DNA	CSA	3.476	11.295	22.681	33.837
	new	0.028	0.031	0.034	0.036
ENGLISH	CSA	6.244	20.244	40.725	58.742
	new	0.031	0.036	0.039	0.042

TABLE II: Index Size for Each Document

Size (KB)		Index
DNA	CSA	30
	new	615
ENGLISH	CSA	50
	new	445

V. Conclusion

In this paper, we have proposed Full-Text Search using Double-Array CDAWG. As described in Section 4, the proposed method can conduct the retrieval much faster than CSA. Future studies will reduce the index sizes, compare the proposed method with other methods for Full-Text Search, and conduct experiments for longer documents.

References

[1] R. Grossi and J. Vitter, "Compressed suffix arrays and suffix trees with applications to text indexing and string matching," in *Proc. the 32nd ACM Symposium on Theory of Computing*, pp. 397–406, 2000.
[2] K. Sadakane, "Compressed text databases with efficient query algorithms based on the compressed suffix array," in *Proc. the 11th International Symposium on Algorithms and Computation*, vol. 1969, pp. 410–421, 2000.
[3] H. W. Huo *et al*., "A practical implementation of compressed suffix arrays with applications to self-indexing," in *Proc. the Data Compression Conference*, pp. 292–301, 2014.
[4] W. K. Hon *et al*., "Practical aspects of compressed suffix arrays and fm-index in searching dna sequences," in *Proc. the 6th Workshop on Algorithm Engineering and Experiments*, pp. 31–38, 2004.

[5] M. Crochemore and R. Vérin, "On compact directed acyclic word graphs," *Structures in Logic and Computer Science*, pp. 192–211, Springer, Verlag, 1997.
[6] S. Inenaga *et al*., "Construction of the CDAWG for a trie," in *Proc. the Prague Stringology Conference '01*, pp. 37–48, 2001.
[7] A. V. Aho *et al*., *Data Structures and Algorithms*, Addison Wesley, 1983.
[8] J. Aoe, "An efficient digital search algorithm by using a double-array structure," *IEEE Transactions on Software Engineering*, vol. 15, no. 9, pp. 1066–1077, 1989.
[9] E. Fredkin, "Trie memory," *Communications of the ACM*, vol. 3, no. 9, pp. 490–499, 1960.
[10] S. Yata *et al*., "Fast string matching with space-efficient word graphs," in *Proc. the 4th International Conference on Innovations in Information Technology*, pp. 79–83, 2008.
[11] R. Gonzâlez *et al*., "Practical implementation of rank and select queries," in *Proc. the 4th Workshop on Experimental and Efficient Algorithms*, pp. 27 38, 2005.
[12] K. Sadakane. (2010). csalib100810. [Online]. Available: http://researchmap.jp/sada/csalib/
[13] Pizza&Chili Corpus. (2005). [Online]. Available: http://pizzachili.dcc.uchil e.cl/

Yuma Fujita received B.Sc. degrees in information science and intelligent systems from Tokushima University, Japan, in 2016. He is currently a M.Sc. student at Tokushima University. His research interests are double-array structures.

Yoshiaki Ichihashi received B.Sc. and M.Sc. degrees in information science and intelligent systems from Tokushima University, Japan, in 2014 and 2016, respectively. He currently works for Toshiba Solutions Corporation.

Shunsuke Kanda received B.Sc. and M.Sc. degrees in information science and intelligent systems from Tokushima University, Japan, in 2014 and 2016, respectively. He is currently a Ph.D. student at Tokushima University. He is a student member of the Information Processing Society in Japan. His research interests are data structures for string processing and indexing.

Kazuhiro Morita received B.Sc., M.Sc., and Ph.D. degrees in information science and intelligent systems from Tokushima University, Japan, in 1995, 1997, and 2000, respectively. He had been a research assistant from 2000 to 2006 in information science and intelligent systems, Tokushima University, Japan. He is currently an associate professor in the Department of Information Science and Intelligent Systems, Tokushima University, Japan. His research interests are sentence retrieval from huge text databases, double-array structures and binary search tree.

Masao Fuketa received B.Sc., M.Sc., and Ph.D. degrees in information science and intelligent systems from Tokushima University, Japan, in 1993, 1995, and 1998, respectively. He had been a research assistant and an associate professor from 1998 to 2000 and from 2000 to 2015 in information science and intelligent systems, Tokushima University, Japan. He is currently a professor in the Department of Information Science and Intelligent Systems, Tokushima University, Japan. He is a member of the Information Processing Society in Japan and the Association for Natural Language Processing of Japan. His research interests are information retrieval and natural language processing.