Using Frequent Substring Mining Techniques for Indexing Genome Sequences: A Comparison of Frequent Substring and Frequent Max Substring Algorithms

Todsanai Chumwatana
College of Information and Communication Technology, Rangsit University, Thailand
Email: Todsanai.c@rsu.ac.th

***Abstract*—The amount of electronically stored information in genome sequence database has grown rapidly in the last decade. This makes frequent substring extraction an essential task as most frequent substrings are meaningful in genome sequences, in order to support the application in the area of information retrieval and data analytics. In this paper, two frequent substring mining techniques are investigated: frequent substring and frequent max substring mining algorithms. Many research communities have acknowledged that the frequent substring mining is one of the viable solutions for extracting the interesting patterns in genome or protein in area of bioinformatics. Beside this, the frequent max substring technique has been proposed as an alternative method to extract meaningful patterns. In this paper, experimental studies and comparison results are shown in order to compare two techniques. From the experimental results, the following observations can be made. The frequent max substring mining technique provides significant benefits over the frequent substring mining technique in term of storage space. Meanwhile, the frequent substring mining technique requires less computational time as this technique is straight forward.**

***Index Terms*—frequent max substring mining, frequent substring mining, genome sequence, frequent max substrings, frequent substrings**

I. Introduction

Over the last decade, genome sequence databases have grown rapidly and have been widely used by molecular biologists for homology searching. The survey shows that the GenBank contains over 77 Gbp (giga, i.e. 10^9, base-pairs) from over 73 million sequence entries [1]. Due to the large amount of data available, the task of providing efficient frequent substring extraction has become important. It has become critical to develop scalable data management techniques for sequence storage, analytic and retrieval. In searching such databases, frequent substring mining techniques are essential for extracting frequent substrings from a massive amount of sequence data for retrieval. This is because the frequent substrings can be treated as index-terms in bioinformatics area. In fact, various algorithms and data structures on strings can be applied to genome sequences because they can be regarded as a sequence of string [2], [3]. However, it is sometimes difficult to use these existing methods for genome sequence databases because of the drawbacks of index sizes.

In this paper, the frequent substring and frequent max substring mining techniques are applied to genome sequencing problems as these two techniques aim to reduce the index size by extracting only frequent substrings. To demonstrate that the frequent substring and frequent max substring mining techniques can be applied to genome sequencing, the experimental and comparison results are presented in this paper. Before the illustration is presented, the characteristics of the genome sequence are first described in the next section, followed by some related works.

II. Characteristic of the Genome Sequence

Human PIPSL	TCACCTCTAG
Chimp PIPSL	TCACCTGTAG
Human	TCACCTCTAG
Chimpanzee	TCACCTCTAG
Cow	TCACCTGTAG
Mouse	TCACCTGTAG
Dog	TCACCTGTAG

Figure 1. Example of nucleotide structure of some species' genes

In the modern era of molecular biology, the genome sequence can be refer to all of a living thing's hereditary information [4]. This hereditary information is encoded in DNA or RNA, which are used for maintaining, building and running an organism, and passing life on to the next generation. In most organisms, the genome includes genes that are packaged in chromosomes, and the non-coding sequences of the DNA that affects specific characteristics of living things. The genome term was introduced by Hans Winkler, Professor of Botany at the University of Hamburg, Germany, in 1920. This genetic

Manuscript received September 19, 2015; revised April 20, 2016.

© 2016 J. Adv. Inf. Technol.
doi: 10.12720/jait.7.4.281-286

material or DNA can be represented as long texts with a specific alphabet, known as the nucleotide bases, for example, {A, C, G, T} in the genome. Most patterns usually occur frequently in the texts because there is only a four-character alphabet to represent genome sequences. A typical example of the genome sequence is shown in Fig. 1.

In fact, the genome contains many relationships. For instance, the genome is the largest part that can be divided into chromosomes, chromosomes are the smaller parts that contain genes, and inside the genes represents the DNA, which is the smallest part. These relationships can be depicted as shown in Fig. 2.

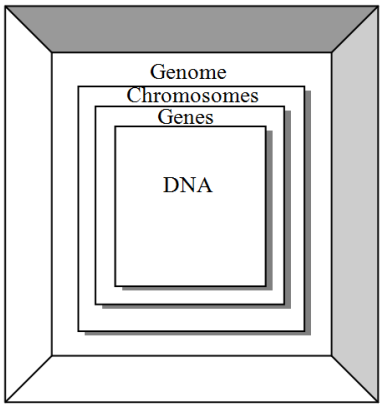

Figure 2. Relationships of genome

There are many types of living things in the world that can be divided into many species such as cows, dogs, mice, chimpanzees, humans and so on. These species have their own distinctive genome. Therefore, genomes can be classified according to species, and can also be used to identify individuals. For example, the genome of people in this world can be classified as the human genome, and each person also has a unique genome and characteristics that can be used to identify individuals. However, two persons may have the same genome if they are identical twins. This significantly shows that the genomes between two persons can be more similar than the genomes between people and other species.

III. Literature Reviews

As mentioned in the first section, genome sequence databases are increasing in size exponentially. Due to this challenge of the over increasing data available, many approaches have been proposed for extracting index terms, indexing and searching from genomic databases. The basic methods proposed earlier perform a full text search without using indices [5]. However, one of the drawbacks of this technique is its poor searching ability. As a result, the suffix tree, suffix trie and suffix array data structures have been widely used in biological sequence analysis, because these structures are fundamental data structures for string matching [6], [3], [1]. Unfortunately, the existing basic algorithms for constructing these data structures do not support large inputs when they are used in real-life applications, thus requiring that the input is small enough to be kept in main memory. Therefore, it is difficult to use them for genome-scale databases, because of their massive amount of index sizes. In order to address this particular drawback, many researchers have improved several algorithms based on these data structures in order to handle huge amounts of genome sequence data.

Vilo introduced an algorithm for discovering frequent substrings from biosequences in 1998 [7], [8]. This algorithm systematically generates a pattern trie while maintaining information about the occurrences of each substring. It is basically a generalization of the *wotd* (write-only top-down) suffix trie construction algorithm [9], [10] to find frequent substrings of a string. This technique is interested in substrings that occur at least at the frequency threshold value in the string, by constructing only the subtrees of the suffix trie that correspond to the frequent substrings. This algorithm has been successfully used for analyzing the full genome of yeast and for predicting certain regulatory elements.

According to [1], [11], Phoophakdee and Zaki proposed an approach for indexing genome-scale sequences using suffix trees, called TRELLIS+, which effectively scales a large amount of genome sequence data using only a limited amount of main-memory, based on a string buffering strategy. Their works focus on a disk-based suffix tree to develop scalable data management techniques for retrieval, analysis and storage of complete and partial genomes. In this algorithm, the index size is not increased when the input sequence is very large. The experimental results showed that TRELLIS+ outperforms existing suffix tree approaches. Their technique was able to index genome-scale sequences and also allowed rapid searching over the disk-based index.

Hugh E. Williams and Justin Zobel proposed a technique for searching genome sequence databases in 2002, known as the index-based approach for both selecting sequences that display broad similarity to a query and for fast local alignment [12]. Several criteria were applied to satisfy the use of this technique. These indexing and retrieval techniques are embodied in a full-scale prototype retrieval system, CAFÉ, that is based on techniques used in text retrieval and in approximate string matching for databases [13]. The principal features of CAFÉ are the incorporation of data structures for query resolution and the indexing technique used. The experimental studies show that this index-based searching technique provides good results with low computational requirements for local alignments. The index-based searching technique produces results which are comparable with existing exhaustive search schemes.

In 2009, Marina Barsky, Ulrike Stege, Alex Thomo and Chris Upton proposed the external-memory suffix tree construction algorithm for very large inputs, known as B^2ST. This algorithm is able to construct suffix trees for input sequences significantly larger than the size of the available main memory [14]. B^2ST minimizes random access to the input string and accesses the disk-based data structures sequentially. It is able to build a disk-based suffix tree for virtually unlimited sizes of input strings,

© 2016 J. Adv. Inf. Technol.

thus filling the ever growing gap between the increase of main memory in modern computers and the much faster increase in the size of genomic databases.

IV. Frequent Substring Mining Technique for Genome Sequences

In this section, the details of two techniques for extracting frequent substrings as index-terms from genome sequences: the frequent substring and frequent max substring mining techniques are reviewed.

A. Frequent Substring Mining Algorithm

Jaak Vilo presented an algorithm for discovering frequent substrings in a string [8]. This algorithm aims at finding substrings that occur frequently in the string (at or above the given frequency threshold value). This is achieved by constructing a pattern trie, which is based on the suffix trie, while maintaining information about the occurrences of each substring. The algorithm constructs only a subtree of the suffix trie that corresponds to frequent substrings of the string, to avoid enumerating the complete set of substrings and in order to reduce the space requirement. It builds the pattern trie for the input string in the breadth-first order, level by level, and creates a list of occurrences for each frequent substring in the string. Frequent substrings are constructed incrementally by expanding prefixes of the substrings that occur at least at the frequency threshold value. Only substrings that occur in the string and occur at least at the frequency threshold value are generated and analyzed. This algorithm has been successfully used for analyzing the full genome of yeast and for predicting certain regulatory elements, and it has also been used for string matching in bioinformatics where the string is a DNA sequence [8].

Vilo's technique is interested only in substrings that occur at least at threshold θ times in the string. It seems that it is not necessary to construct the subtrees with less than threshold θ leaves. As a result, Vilo's algorithm only builds the part that contains frequent substrings. The algorithm is based on the suffix trie data structure. The construction procedure is inspired by the lazy algorithm [9] for generating a suffix trie. The algorithm is a generalization of the *wotd* (write-only top-down) suffix trie construction algorithm, to find the frequent substrings of a string. The resulting trie contains all frequent substrings. The nodes of the trie are labeled with the substrings. Labels on the path from the root to an internal node form the substring associated with that node. Thus each internal node represents a substring of the string and each terminal node represents a suffix of the string. The trie is called the pattern trie in Vilo's algorithm.

At each node, an occurrence list is maintained that contains the position of each occurrence of the substring corresponding to the node. The trie is generated starting from the root. The root corresponds to the empty pattern λ the occurrence list of which contains all character positions of the string. The trie is extended by generating the nodes in the trie in a systematic way. At each step, the children of some of the current leaf nodes are generated and inserted into the trie to make new leaf nodes. For a node N with associated substring ABC, every legal extension ABCD is generated by inserting a new child with label D under the node N. The occurrence list of ABCD is computed from the occurrence list of ABC by checking for each occurrence of ABC in the string to see if it can be extended to an occurrence of ABCD.

Each node N in the trie can be identified by the substring x that is the sequence of labels along the path from the root to the node N. This node N can be denoted by $N(x)$. Hence, N(ABC) is the node identified by substring ABC, and $N(x\text{D})$ is the child of $N(x)$ with character label D that equals N(ABCD). Every node in the trie contains additional information about its relation to other nodes in the tree. The dot-notation will be used to represent subfields—for example *N.parent*, *N.child*, *N.char* and *N.sibling*. The substring x is formed by the character labels *N.char* along the path from the root to the node $N(x)$, $N(x\text{D}).char = a_i$ that is the character label D where $a_i \in \sum$, and $N(x\text{D}).parent = N(x)$. Given the node N, $N.child(a_i)$ is used to denote the child P of node N so that $P.char = a_i$. A sibling of node N can be identified by the shorthand notation $N.sibling(a_i)$, where $N.sibling(a_i)$ is actually $N.parent.child(a_i)$. Note that $N.sibling(a_i)$ is the same as N if $N.char = a_i$. To keep the information about the occurrences of each substring, the lists of character positions of the string where the substring occurs are used. The occurrence list of substring x is stored in the node $N(x)$ and denoted by $N(x).pos$. In addition, the frequency of substring x, $f_s(x)$, can be calculated from the number of substring positions.

Vilo's algorithm starts by building the suffix trie for the input string s in a systematic order, for example in the breadth-first order, level by level. For each node $N(x)$ create the list of positions $N(x)$. *pos* containing each location of the string s where x occurs. To represent the occurrence that ends at character position j of the string s, a pointer is used to position j+1. To create the children of node $N(x)$, find characters $a_i \in \sum$ for which the substring xa_i occurs in at least at θ different locations of the string s. This corresponds to counting which characters of $\sum$ occur at least threshold θ at the positions $N(x)$. *pos* of the string s. This can be done by one traversal of the position list $N(x).pos$ and creating simultaneously all the position lists for every character occurring at these positions in the string s. Only these nodes $N(xa_i)$ are inserted into the trie, for which the character a_i occurs at least threshold θ at positions $N(x).pos$.

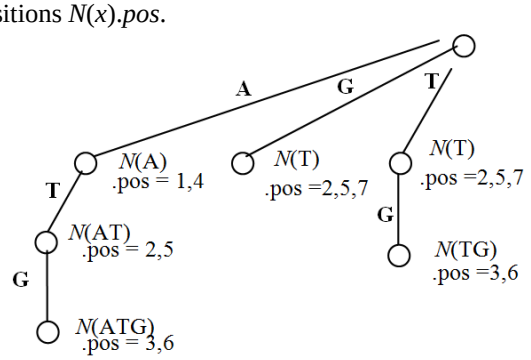

Figure 3. Discovering frequent substrings of genome sequence S = 'ATGATGT' having at least two occurrences.

© 2016 J. Adv. Inf. Technol.

The trie is constructed by systematically extending the leaf nodes. Thus, the position lists are needed only for the leaves during the trie construction. An example of trie construction in discovering frequent substrings from the genome sequence is depicted in Fig. 3.

Let genome sequence S = 'ATGATGT' and θ =2

Sequence S = A T G A T G T

pos = 1 2 3 4 5 6 7

N (λ).pos = 1, 2, 3, 4, 5, 6, 7

In Vilo's algorithm, each node in the trie represents a unique substring and contains the list of positions in the string where the substring occurs. From Fig. 3, the algorithm generated six frequent substrings from string s as shown in Table I.

TABLE I. ALL FREQUENT SUBSTRING WITH NUMBER OF OCCURRENCE

Substring	Number of occurrences
T	3
A	2
G	2
AT	2
TG	2
ATG	2

The strength of Vilo's algorithm is that this technique requires less storage space and construction time than the suffix tree and suffix trie for indexing the frequent substrings when θ > 1. This is because the algorithm constructs only subtrees of the suffix trie that correspond to the frequent substrings to avoid enumerating all substrings.

B. Frequent Max Substring Mining Algorithm

Frequent max substring mining technique is based on text mining that describes a process of discovering useful information or knowledge from unstructured texts [15], [16], [17]. This technique is used to classify index-terms called frequent max substrings from genome sequences where the word boundaries are not clearly defined. The frequent max substrings refer to the substrings that appear frequently (at a predetermined frequency *f*) and have the maximum length of n-grams on the given string, so these terms are likely to be the patterns of interest. The set of frequent max substrings is also able to contain all frequent substrings which appear on the given sequences.

In order to explain the concept, the following shows the process of the frequent max substring mining technique using Min Heap and reduction rules to extract the frequent max substrings as index-terms from genome sequences.

Let genome sequence S = 'ATGATGT'

Position(.pos) = 1 2 34 567

and predetermined frequency f = 2

Min-heap structure

Firstly, all substrings with a length of 1 are extracted, together with their frequencies and list of positions. The frequencies of these substrings are then checked in order to select only the frequent substrings with a length of 1. These frequent substrings are finally kept in the min-heap structure for further processes.

A, 2 .pos=1, 4	T, 3 .pos=2, 5, 7	G, 2 .pos=3, 6

Next, <A, 2> is removed from min-heap in order to indicate that <A, 2> is detected and extracts its child substrings for the next process. After <A, 2> is removed from min-heap, the algorithm extracts child substrings of <A, 2> using list of positions or pointers of <A, 2> to reduce time complexity. Child substrings consist of <AT, 2>. <AT, 2> is kept in min-heap using the insertion rule, because <AT, 2> is the substring that occurs in two different positions in string s.

T, 3 .pos=2, 5, 7	AT, 2 .pos=2, 5	G, 2 .pos =3, 6

<T, 3> is removed from min-heap, after which child substrings of <T, 3> are extracted using the list of positions or pointers of <T, 3>. Child substrings consisting of <TG, 2> and <T$, 1>. <G, 2> are deleted from min-heap because <TG, 2> is a proper superstring of <G, 2> at the same frequency, and <TG, 2> is kept in min-heap instead, using the insertion rule, because its frequency is equal to the predetermined frequency.

AT, 2 .pos=2, 5	TG, 2 .pos =3, 6

<AT, 2> is removed from min-heap and then its child substrings are extracted using its list of positions (pointers). They consist of <ATG, 2>. <TG, 2> is deleted from min-heap because <TG, 2> is a substring of <ATG, 2> with the same frequency. After that, <ATG, 2> is kept in min-heap using the insertion rule because <ATG, 2> is the substring that occurs in two different locations in string s.

ATG:2 .pos =3, 6	

<ATG, 2> is removed from min-heap and then its child substrings are extracted using its list of positions. They consist of <ATGA, 1> and <ATGT, 1>. They are not kept in min-heap because their frequencies are less than predetermined frequency.

The algorithm will stop when min-heap is empty. This means all substrings in min-heap were detected and processed completely.

© 2016 J. Adv. Inf. Technol.

From the above algorithm, the resulting trie can be depicted in Fig. 4.

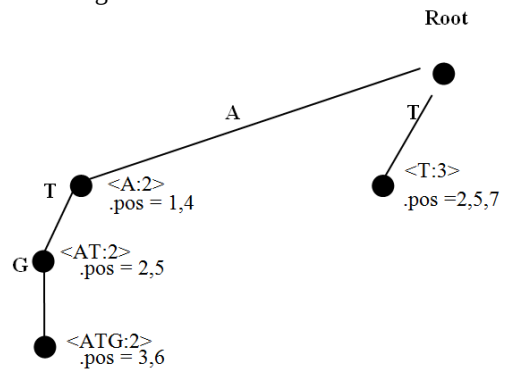

Figure 4. Frequent suffix trie structure using proposed frequent max substring technique.

Fig. 4 shows the result tried from frequent max substring mining technique. From observation, the frequent max substring set is able to contain all frequent substrings as shown in Table II.

TABLE II. NUMBER OF FREQUENT MAX SUBSTRINGS AND FREQUENT SUBSTRINGS

Frequent max substrings	Frequent substrings
<T, 3>	<T, 3>
<ATG, 2>	<A, 2> <G, 2> <AT, 2> <TG, 2> <ATG, 2>

It can be observed from Table II that <T, 3>, <A, 2>, <G, 2>, <AT, 2>, <TG, 2> and <ATG, 2>, which extracted from the frequent substring mining technique, are substrings of <T, 3> and <ATG, 2>, which are extracted from the frequent max substring mining technique. The indexing terms <A, 2>, <G, 2>, <AT, 2>, <TG, 2> and <ATG, 2> can be enumerated from indexing term <ATG, 2>, while <T, 3> can be enumerated from <T, 3>. It is considered that <T, 3> and <ATG, 2> can be kept, instead of keeping <A, 2>, <G, 2>, <AT, 2> and <TG, 2> in order to reduce index size and the number of index- terms.

V. EXPERIMENTAL STUDIES AND COMPARISON RESULTS

In this section, the experiment and comparison result of extracting frequent substring as index-terms from genome sequences using the frequent substring and frequent max substring mining techniques are presented. In order to compare two different algorithms for genome sequencing, the number of index-terms that are extracted by using frequent substring and the frequent max substring mining techniques are compared. The dataset used for this evaluation is the genome sequence found on the website: http://www.broadinstitute.org/cgi-bin/annotation/methanosarcina/download-sequence.cgi. Genome sequences have various lengths. The set of genome sequences consists of 20 sequences and contains 52,500 characters. The sequence lengths start from 250 to 5,000 characters.

The results showed that frequent substring mining technique, Vilo's algorithm, extracted a greater number of index-terms than the frequent max substring mining technique at the low given frequency threshold value. All index-terms extracted from Vilo's algorithm were contained by all index-terms extracted from the frequent max substring mining technique as described in previous section. However, the number of index-terms is likely to be closer to each other when the given frequency threshold value increases. The reduction rate can be evaluated using the measurement in the proportion of the number of index-terms extracted from two techniques as shown in Fig. 5.

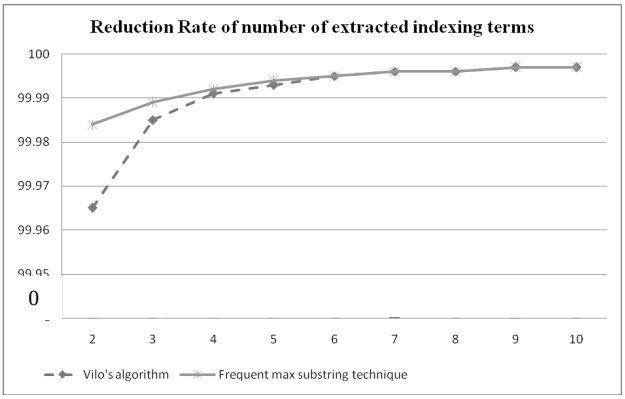

Figure 5. Reduction rate of number of index-term enumerations using frequent max substring technique and frequent substring technique (Vilo's technique).

From Fig. 5, the experimental results showed that the frequent max substring mining technique can increase the reduction rate of the number of indexing terms by up to 99.98 percent at low frequency threshold value, and the reduction rate has increased slightly at the higher frequency threshold value. Meanwhile, the reduction rate of Vilo's technique is lower than the reduction rate of the frequent max substring mining technique, although Vilo's algorithm also provided a high reduction rate of up to 99.96 percent at low frequency threshold value. However, it can be observed that the reduction rate of both algorithms is likely to be closer when the frequency threshold value is higher. In addition, the reduction rate of the number of index-terms for both algorithms increases according to the given frequency threshold values and the maximum size of index-terms. The comparative study of both approaches in term of runtime complexity are also revealed. From the experimental studies, Frequent max substring mining technique takes $O(n^2)$ time complexity, where n is the number of characters in sequence. Meanwhile, Frequent substring mining technique takes $O(nd)$ time complexity where d is the size of maximum index-terms. This shows that

© 2016 J. Adv. Inf. Technol.

Frequent max substring mining technique spends more time than Frequent substring mining technique. However, the indexing process is regarded as the backend process that can be run behind the scene.

VI. Conclusion

This paper describes and compares two frequent substring mining techniques: the frequent substring mining (Vilo' algorithm) and frequent max substring mining techniques for genome sequences. These two techniques are regarded as a viable solution for extracting frequent substrings as index-terms in genome sequence databases and also used in the area of bioinformatics. From the experimental studies and comparison results, the frequent max substring mining technique provides significant benefits over the frequent substring mining technique in term of storage space. This is because the frequent max substring mining technique can increase the reduction rate of the number of indexing terms by up to 99.98 percent at low frequency threshold value, and the reduction rate has increased slightly at the higher frequency threshold value. Meanwhile, the reduction rate of Vilo's technique is lower than the reduction rate of the frequent max substring mining technique.

References

[1] B. Phoophakdee and M. J. Zaki, "TRELLIS+: An effective approach for indexing genome-scale sequences using suffix trees," in *Proc. Pacific Symposium on Biocomputing*, Kohala Coast, Hawaii, USA, 2008.

[2] H. Huo and V. Stojkovic, "A suffix tree construction algorithm for DNA sequences," in *Proc. the 7th IEEE International Conference on Bioinformatics and Bioengineering*, Boston, 2007, pp. 1178-1182.

[3] K. Sadakane and T. Shibuya, "Indexing huge genome sequences for solving various problems," *Genome Informatics*, vol. 12, pp. 175-183, 2001.

[4] P. Baldi and G. W. Hatfield, *DNA Microarrays and Gene* Expression*: From Experiments to Data Analysis and Modeling*, United Kingdom: Cambridge University Press, 2002.

[5] D. E. Knuth, J. H. Morris Jr, and V. R. Pratt, "Fast pattern matching in strings," *SIAM Journal on Computing*, vol. 6, no. 2, pp. 323-350, 1977.

[6] M. Barsky, U. Stege, A. Thomo, and C. Upton, "Suffix trees for very large genomic sequences," in *CIKM '09 Proc. the 18th ACM Conference on Information and Knowledge Management*, 2009.

[7] J. Vilo, "Pattern discovery from biosequences," Facutly of Science, Department of Computer Science, PhD thesis, University of Helsinki, Helsinki, November 2002.

[8] J. Vilo, "Discovering frequent patterns from strings: Department of computer science, university of Helsinki, Finland," *Technical Report C-1998-9*, May 1998.

[9] R. Giegerich and S. Kurtz, "A comparison of imperative and purely functional suffix tree constructions," *Science of Computer Programming*, vol. 25, no. 2-3, pp. 187-218, 1995.

[10] R. Giegerich, S. Kurtz, and J. Stoye, "Efficient implementation of lazy suffix trees," in *Proc. the Third Workshop on Algorithmic Engineering (WAE99)*, 1999, pp. 30-42.

[11] B. Phoophakdee and M. J. Zaki, "Genome-scale disk-based suffix tree indexing," in *Proc. the ACM SIGMOD International Conference on Management of Data*. 2007, pp. *833-844*.

[12] H. E. Williams and. J. Zobel, "Indexing and retrieval for genomic databases," *IEEE Transactions on Knowledge and Data Engineering*, 2002, pp. 63-78.

[13] H. E. Williams, "Indexing and retrieval for genomic databases," Ph. D thesis, RMIT University, Melbourne, Australia, 1998.

[14] M. Barsky, U. Stege, A. Thomo, and C. Upton, "A new method for indexing genomes using on-disk suffix trees," in *Proc. CIKM '08:17th ACM Conference on Information and Knowledge Management*, 2008, pp. 649-658.

[15] T. Chumwatana, "Genome sequence clustering using hybrid method: Self-organizing map and frequent max substring techniques," in *Proc. the 12th International Conference on Machine Learning and Cybernetics*, Tianjin, China, 2013.

[16] T. Chumwatana, K. W. Wong, and H. Xie, "Frequent max substring mining for indexing," *International Journal of Computer Science and System Analysis.*

[17] T. Chumwatana "Using N-gram and frequent max substring techniques for index-term extraction from non-segmented texts: A comparison of two techniques," *The Journal of Information Science and Technology*, vol. 3, no. 1, 2012.

Asst. Prof. Dr. Todsanai Chumwatana– Associate Dean for Academic of Information and Communication Technology, Rangsit University, Pathumtani, Thailand. His major fields of scientific research contain Machine Learning Text Mining, Natural Language Processing, Artificial Intelligence, Medical Informatics.

© 2016 J. Adv. Inf. Technol.