An External Memory Approach to Compute the Statistics of Maximal Repeats from Whole Genome Sequences

從整個基因體序列中, 一個利用外部記憶體去計算最大重複統計資料的方法

Jing-Doo Wang
Department of Computer Science and Information Engineering
Asia University
Taiwan, R.O.C.
jdwang@asia.edu.tw

Abstract

The objective of this paper is to develop an external memory approach to extract the maximal repeats from whole genome sequences with the statistics of these repeats across classes, where the definition of a class is determined on what kind of statistics one wants to compute. We proposed a heuristic method consisted of a bucket-sort-like approach and the Chinese term extraction approach. The former was used to sort the suffixes of DNA sequences stored in files and the later was used to extract maximal repeats by scanning the sorted suffixes while computing the statistics of these repeats. The statistics of these repeats across classes might be useful for sequence classification and species identification.

Keywords: maximal repeat; external memory; comparative genomics.

摘要

這篇論文利用外部記憶體的方法, 整個基因體序列中, 求最大重複在各類別中的統計資料, 其中類別的定義是根據你所需要的統計資料而定。我們所提出的啓發式的方法是由一種類似水桶式排序法與中文關鍵詞抽取法所組成。前者是用來排序儲存在檔案中 DNA 序列的 suffixes, 後者利用逐一掃瞄排序過的 suffixes 的時候, 抽取出最大重複, 並且同時計算出這些重複的統計資料。這些跨類別的最大重複統計資料, 可能對於序列分類與物種鑑定有幫助。

關鍵詞: 最大重複、外部記憶體、比較基因體。

1 Introduction

Nowadays the growth of computation power might be far beyond the need for biologists to handle the whole genome sequences comparison among organisms simultaneously. One of the bottlenecks of the computation for comparative genomics is due to the limit of main memory available in one general computer such that the sizes of data sets could not exceed the capacity of main memory of that computer.

The statistics of repeats across classes might be useful for inter-genomic comparison [14, 25, 20, 6] because we might have one species as one class to select the repeats with biased frequency distribution for sequence classification[17], or to mine for specific patterns to identify some species for microarray-based detection [2, 7]. However the size of individual whole genome sequence might range from several million base pairs to hundreds billion base pairs. The internal memory approaches, for example, using suffix tree or suffix array[9] are not applicable to the comparison of whole genome sequences[3] with a general computer. Therefore, it might take a very long time to finish the computation when the amount of memory needed to process sequences comparison are over the capacity of the main memory in that computer[18] because of the disk I/O problems[24].

The analysis of repeats plays an important role for comparative genomics. There are studies of repeats extraction such as STRING [19], REPuter [12], MUMmer [5], FORRepeats [13] and SRF[23]. STRING is used to find tandem repeats in DNA sequences via alignment. FORRepeats is a heuristic approach using a data structure called factor oracle to find repeats. Sharma et al. used Spectral Repeat Finder (SRF), a discrete Fourier transformation, to identify repetitive DNA sequences. REPuter and MUMmer were based on the suffix tree to detect repeats. However, suffix tree suffered from the large memory consumption although there were approaches to reduce the memory consumption for constructing the suffix tree[1, 11, 16].

Choi and Cho [3] proposed a workbench called *SequeX*, an external memory approach, which based on Static SB-tree (SSB-tree) [8] to analysis of common k-mers for whole genome sequences from 72 microbial genomes. The *SequeX* supported queries such as "Pattern Matching", k-mer Counting", and "Finding k-Occurring Sequence". However, it was not trivial to generate the program of constructing the SSB-tree because it needs efficiently to handle the disk accesses and to build the blind tree for determining the next node for searching.

In this study, we used the maximal repeat [9] to reduce the redundancy of repeats such that do not generate overwhelming output. A maximal pair [12] in a string S is a pair of identical substrings α and β in S such that the character to the immediate left, resp. right, of α is different from the character to the immediate left, resp. right, of β. That is, extending α and β in either direction would destroy the equality of the two strings. A maximal repeat α is a substring of S that occurs in a maximal pair in S. Given a sequence "kabcyiazabczabcyrxak", for example, the patterns, "abc" and "abcy" are maximal repeats, while "ab" and "bc" are not.

The objective of this work is to extract the maximal repeats from DNA sequences while computing the statistics across classes with limited memory, e.g. 2 GB main memory, of one general computer where the definition of a class is determined on what kind of statistics one wants to compute in advance. Our approach is simple and straightforward, and could be easily scaled to parallel processing. To extract the maximal repeats and to compute the statistics of those patterns across classes, we proposed a heuristic method consisted of a bucket-sort-like approach[4] and the Chinese term extraction approach [28]. The former was used to sort the suffixes of DNA sequences and the later was used to extract maximal repeats while computing the statistics of these repeats across

classes. To handle such a huge amount of DNA sequences across classes with limited memory, first of all, we pre-sort the suffixes of DNA sequences by partitioning these suffixes into a given number 4^k of groups (files), where k is the length of the common prefix of the suffixes in one group. Then we sorted the suffixes in each group individually. After sorting all the suffixes in each group, we used the scanning process of the Chinese term extraction approach to extract the candidate maximal repeats with the right boundary verification. To have the candidate maximal repeats with the left boundary verification, we could rerun the above processes with the suffixes of reversed DNA sequences. The maximal repeats are those candidate maximal repeats which pass both the right and the left boundary verification. Note that we could easily speed up the computation by assigning the jobs of sorting suffixes and extracting candidate maximal repeats to different processors according to the value k.

We had experimented with six species, including "SARS", "E.coli", "Yeast", "P. falciparum", "C. elegant" and "A. thaliana". The lengths of these DNA sequences ranged from several thousand (bp) to over 100 million (bp). In practice, we had the suffixes as the front l, e.g. $l = 50$, characters of every suffixes instead of the whole suffixes; the value of k was determined manually in this study such that the sorting of the suffixes in one group could be done efficiently with limited memory. That is, the length of maximal repeats extracted by our approach is between k and l, where k is the common prefix of suffixes in one group (file), and l is the length of the front characters of suffixes used. To evaluate the variant values of k and l, we had experimented with the values of k including l, 2, 3 and 4, the value l of l including 25, 50, 100, 200, 500 and 1000. Note that the value 1000 of the length of maximal repeats was much longer than that of patterns extracted by the general slide-window methods [10] or that for primers design [22]. To show the scalability of our approach for parallel computing, furthermore, we also presented the speedup via 8 computers where each contained 512 MB main memory. The value of speedup we achieved was 4.94 when $k = 4$ and $l = 50$ using the Yeast's genome.

The remainder of this paper is organized as follows. Section 2 describes the processes of maximal repeat extraction. Section 3 gives experimental results. Section 4 gives conclusions and discussions.

2 Methods

First of all, we give the definition of maximal repeat as follows [9, 12].

Definition 1 (Maximal Repeat) *Consider a sequence S over the DNA alphabet. A repeat is a substring in S which occurs at least twice. Support w is a repeat of length l in S which occurs at the starting positions i and j, i.e. $S_i \cdots_{i+l-1} = S_j \cdots_{j+l-1}$. We have w as a* maximal repeat *if $S_{i-1} \neq S_{j-1}$ and $S_{i+l} \neq S_{j+l}$.*

Our approach first used a bucket-sort-like approach[4] to sort the suffixes lexicographically, and then modified the Chinese term extraction approach [28] to extract the candidate maximal repeats while computing the statistics of these repeats across

classes. Finally, we had the maximal repeats as those candidate maximal repeats which passed both the right and the left boundary verification. We described the details in section 2.1, section 2.2 and section 2.3, respectively.

2.1 Sorting the Suffixes

Due to the alphabet of DNA sequences is limited to $\{A, C, G, T\}$, we can partition the suffixes strings into groups (files) according to their prefix, and then sort the strings in one group (file) individually.

First of all, all the suffixes of DNA sequences were partitioned into 4^k groups (files) according to the prefix of suffixes, where k is the length of the common prefix. Note that the value of k can be determined manually such that the amount of suffixes in the largest group could be processed efficiently with limited memory . For example, as shown in Fig. 1, there are the suffixes of the sequence "ACTTTCACTTCCGGCAATTAGCCGATTTTC" whose length is 30. As shown in Fig. 2, the value of k is 1 and the number of groups (files) is $4^1 = 4$. Secondly, those suffixes in one group could be sorted individually in lexicographic order by internal-sorting methods[9] or by the external-sorting methods [3, 26]. The suffixes of DNA sequences and revered DNA sequences are individually sorted for determining the right and the left boundaries of maximal repeats, respectively. Figure 3 shows the sorted suffixes when $k = 1$. In this study, for simplicity, we had the front l characters of suffixes instead of the whole suffixes such that the length of maximal repeats ranged from k to l.

Position	Suffix String
1	ACTTTCACTTCCGGCAATTAGCCGATTTTC
2	CTTTCACTTCCGGCAATTAGCCGATTTTC
3	TTTCACTTCCGGCAATTAGCCGATTTTC
4	TTCACTTCCGGCAATTAGCCGATTTTC
5	TCACTTCCGGCAATTAGCCGATTTTC
6	CACTTCCGGCAATTAGCCGATTTTC
7	ACTTCCGGCAATTAGCCGATTTTC
8	CTTCCGGCAATTAGCCGATTTTC
9	TTCCGGCAATTAGCCGATTTTC
10	TCCGGCAATTAGCCGATTTTC
11	CCGGCAATTAGCCGATTTTC
12	CGGCAATTAGCCGATTTTC
13	GGCAATTAGCCGATTTTC
14	GCAATTAGCCGATTTTC
15	CAATTAGCCGATTTTC
16	AATTAGCCGATTTTC
17	ATTAGCCGATTTTC
18	TTAGCCGATTTTC
19	TAGCCGATTTTC
20	AGCCGATTTTC
21	GCCGATTTTC
22	CCGATTTTC
23	CGATTTTC
24	GATTTTC
25	ATTTTC
26	TTTTC
27	TTTC
28	TTC
29	TC
30	C

Figure 1: Suffixes examples

	Position	Suffix String
A	1	ACTTTCACTTCCGGCAATTAGCCGATTTTC
	7	ACTTCCGGCAATTAGCCGATTTTC
	16	AATTAGCCGATTTTC
	17	ATTAGCCGATTTTC
	20	AGCCGATTTTC
	25	ATTTTC
C	2	CTTTCACTTCCGGCAATTAGCCGATTTTC
	6	CACTTCCGGCAATTAGCCGATTTTC
	8	CTTCCGGCAATTAGCCGATTTTC
	11	CCGGCAATTAGCCGATTTTC
	12	CGGCAATTAGCCGATTTTC
	15	CAATTAGCCGATTTTC
	22	CCGATTTTC
	23	CGATTTTC
	30	C
G	13	GGCAATTAGCCGATTTTC
	14	GCAATTAGCCGATTTTC
	21	GCCGATTTTC
	24	GATTTTC
T	3	TTTCACTTCCGGCAATTAGCCGATTTTC
	4	TTCACTTCCGGCAATTAGCCGATTTTC
	5	TCACTTCCGGCAATTAGCCGATTTTC
	9	TTCCGGCAATTAGCCGATTTTC
	10	TCCGGCAATTAGCCGATTTTC
	18	TTAGCCGATTTTC
	19	TAGCCGATTTTC
	26	TTTTC
	27	TTTC
	28	TTC
	29	TC

Figure 2: Partitioning suffixes into A, C, G and T groups ($k = 1$).

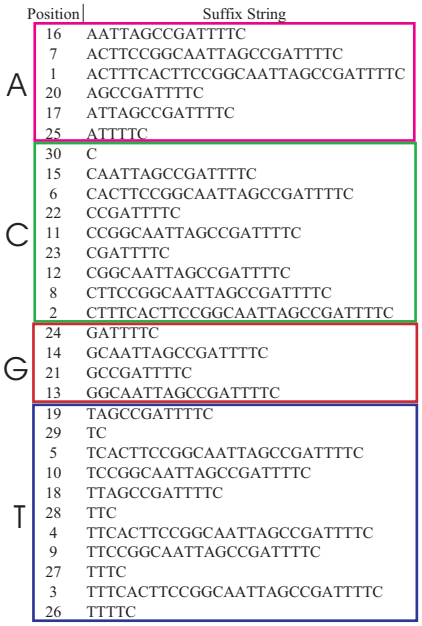


Figure 3: Suffixes sorted in groups ($k = 1$)

2.2 Extracting the Candidate Maximal Repeats

In this study we modified the scanning process of the Chinese term extraction approach [28] to extract the candidate maximal repeats. In brief, we scanned lexicographically the suffixes of DNA sequences, resp. reversed DNA sequences, and had the longest common prefix of adjacent strings as candidate maximal repeats with the right, resp. the left, boundary verification. To make the paper more self-contained, we had an example for the extraction of candidate maximal repeats in Appendix A.

2.3 Verifying the Candidate Maximal Repeats

The maximal repeats are those candidate maximal repeats which pass both the right and the left boundary verification. We first

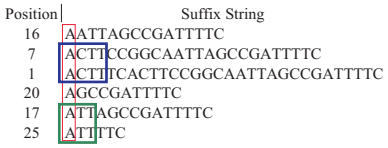


Figure 4: The candidate maximal repeats with right boundary verification

Figure 5: The candidate maximal repeats with left boundary verification

stored all the repeats passed the left boundary verification into hash (or database). Then, we verified each candidate maximal repeat which passed the right boundary verification as one maximal repeat if its reversed string existed in that hash. In practical, the number of maximal repeats might rise exponentially with the size of the genome. therefore, we might have loaded the partial of the repeats passed the left boundary verification into hash and check with the repeats passed the right boundary verification several times.

For example, there were the partial of sorted suffixes whose prefix begin with "A" as shown in Fig. 4, and the patterns "ACTT" and "ATT" were candidate maximal repeats with the right boundary verification. Note that the patterns "AC", "ACT" and "AT" didn't pass the right boundary verification. Because the corresponding reversed patterns, "TTCA" and "TTA", also passed the left boundary verification as shown in Fig. 5, the pat-

terns "ACTT" and "ATT" were maximal repeats.

3 Experimental Results

We had experimented with the DNA sequences of the whole genomes from six species, including "SARS", "E.coli", "Yeast", "P. falciparum", "C. elegant" and "A. thaliana". The experiments were computed using one server with 2*Xeon 2.4 GHz CPU, 2 GB RAM and Red Hat Linux 9.0.

3.1 The Computation Time with Different Species

As shown in Table 1, the lengths of these DNA sequences ranged from several thousand (bp) to over 100 million (bp). The time for computing the statistics of maximal repeats of "A. thaliana", for example, was about 72 hours and 38 minutes and the number of maximal repeats extracted was $124,462,282$. Note that, in this experiment, we also took the negative strand into consideration by concatenating the sequence with its reversed-complement sequence.

3.2 The Computation Time with Different Values of k and l

To evaluate the computation time for the variation of l, first, we fixed the value of k as 1 and performed an experiment with the genome sequences of Yeast according to the different values of l, including 25, 50, 100, 200, 500 and 1000. As shown in Table 2, the total computation time was about 39 hours and 31 minutes when $l = 1000$ and the extraction process took the majority of computation time; the computation time of each process increased as long as the value of l increased. This part of computation could be further improved by sorting the suffixes via the indexes of suffixes in the files instead of sorting the suffixes.

Secondly, as shown in Table 3, we fixed the value of l as 50 and experimented with the different values of k, including 1, 2, 3 and 4. The total computation time kept almost the same as the value of k increased, and somewhat was reduced because the amount of the suffixes in one file decreased that might speed up the process of sorting suffixes. The value of k, however, was limited to under 5 because the default maximal number of opening files using Linux (Red Hat Linux 9.0) was 1024 if we didn't re-compiler the kernel. However, this problem to extend the value of k could be solved using parallel computing.

3.3 Intra-Chromosomal Comparison

To see the frequency distribution of maximal repeats across the 16 chromosomes of yeast, we had each chromosome as one class. Let class frequency (cf) be the number of classes that the pattern appears, and let term frequency (tf) be the number of the pattern appears, and let the length (len) be the length of maximal repeat. Figure 6 showed the statistics of class frequency of maximal repeats with $tf \geq 10$ and $len \geq 30$. Among these maximal patterns, for example, there were 56 patterns appeared just in one chromosome and there were 7 patterns appeared in all the 16 chromo-

Table 1: The statistics of DNA sequences for maximal repeats extraction

Species	Length (bp)	TotalTime (HH:MM:SS)	# of Maximun Repeats
SARS	29,736	0:00:37	32,358
Ecoli	4,639,675	1:51:02	5,021,132
Yeast	12,070,527	11:20:47	12,835,695
P. falciparum	22,820,308	20:16:16	22,821,291
C. elegan	100,096,025	60:01:07	103,730,344
A. thaliana	119,186,497	72:37:48	124,462,282

Table 2: The computation time for the different values of l when $k = 1$ (Yeast)

	The front l characters of suffixes					
Processes	**25**	**50**	**100**	**200**	**500**	**1000**
Partition Suffixes (right boundary)	00:03:58	00:04:28	00:05:22	00:07:50	00:13:57	00:24:30
Partition Suffixes (left boundary)	00:04:21	00:04:25	00:05:27	00:07:40	00:14:00	00:24:48
Sort Suffixes (right boundary)	00:11:44	00:13:59	00:18:18	00:29:04	01:10:26	02:06:16
Sort Suffixes (left boundary)	00:11:32	00:14:07	00:18:24	00:29:28	01:10:21	02:06:05
Extract Candidate Maximal Repeats (right boundary)	03:44:51	04:07:22	04:44:08	06:00:17	09:34:27	15:37:41
Extract Candidate Maximal Repeats (left boundary)	03:47:19	04:07:39	04:45:48	05:57:23	09:37:52	15:23:43
Verify the Candidate Maximal Repeats	02:04:58	02:07:06	02:09:01	02:15:43	02:34:23	03:27:36
Total	10:08:43	10:59:06	12:26:28	15:27:25	24:35:26	39:30:39

Table 3: The computation time for the different values of k when $l = 50$ (Yeast)

	The length k of common prefix			
Processes	k=1	k=2	k=3	k=4
Partition Suffixes (right boundary)	00:04:28	00:04:37	00:04:32	00:04:39
Partition Suffixes (left boundary)	00:04:25	00:04:33	00:04:32	00:04:42
Sort Suffixes (right boundary)	00:13:59	00:13:46	00:12:08	00:11:54
Sort Suffixes (left boundary)	00:14:07	00:13:06	00:12:21	00:11:54
Extract Candidate Maximal Repeats (right boundary)	04:07:22	04:03:36	04:05:58	04:06:09
Extract Candidate Maximal Repeats (left boundary)	04:07:39	04:05:55	04:04:13	04:03:06
Verify the Candidate Maximal Repeats	02:07:06	02:07:43	02:07:08	02:06:29
Total	10:59:06	10:53:16	10:50:52	10:48:53

somes. These observations might be useful to evaluate the intra-chromosomal comparison.

3.4 The Statistics of Maximal Repeats Across Species

It might be interesting for some experts to discovery the distinctive patterns as the class markers if some patterns just appear in one class but do not appear in the other classes. Let class frequency (cf) be the number of classes, species in this study, that the pattern appears, and let term frequency (tf) be the number of the pattern appears, and let the length (len) be the length of maximal repeat. Table 4 showed the statistics of the maximal repeats across six species as we had one species as one class. There were distinctive maximal patterns extracted from the DNA sequences of these species. For example, the pattern "TATAAAATTTTTTTTTTATTTTTT-TATTTTTATTTAAATTTCCATT-TAAT" with $len = 50$ and $tf = 4$ appeared only in the "P. falciparum"; the pattern "TATAAAATTTTTTTTTTTCAAAGTTTC", with $tf = 2$ and $cf = 2$, appeared both in the "C. elegan" and the "P. falciparum". Note that the patterns with $tf = cf$ could not be detected just from one species without making the comparison across species.

4 Conclusions and Discussions

We developed an external memory approach to extract maximal repeat from DNA sequences with the statistics of frequency distribution across classes such that we could have genomic sequences comparison across species possible in the vector space model [21]. It is attractive to select representative repeats (patterns) whose frequency distribution across classes were biased to present sequences (species) as vectors such that many machine learning algorithms [15] in vector space model could be applied to those bioinformatics problems [14], such as sequence classification [17]. In deed, we had this approach to extract the representative patterns for virus classification [27].

4.1 Time Complexity Analysis

Our approach consisted of three main steps, partitioning suffixes into groups (files), sorting suffixes in one group (file), extracting and verifying the candidate maximal repeats. Suppose the total length of DNA sequences is N. The time complexity of our approach was approximated to $O(4^k * N + 4^k * (\frac{N}{4^k})log(\frac{N}{4^k}) + |MR|)$, where k was the length of the common prefix of those strings in one group and $|MR|$ was the number of candidate maximal repeats extracted. The time complexity of the sorting of suffixes in 4^k files was approximated to $O(4^k * (\frac{N}{4^k})log(\frac{N}{4^k}))$ assuming that the suffixes were uniformly distributed in each group using one processor. The partition of suffixes into 4^k groups (files) could be achieved with time complexity $O(4^k * N)$ by scanning the DNA sequences and outputting those suffixes with the same prefix into one file at a time. This part of computation time could be reduced to $O(N)$ by opening 4^k files concurrently and outputting the strings to their corresponding files. The value of k, however, was limited

Table 4: The statistics of the partial maximal repeats across six species (k=4,l=50)

Maximal Repeats	len	tf	cf	Frequency Distribution					
				A. thaliana	C. elegan	E. coli	P. falciparum	SARS	Yeast
TATAAAATTTTTTTTTGG	18	2	2	1	0	0	1	0	0
TATAAAATTTTTTTTTG	17	5	3	1	3	0	1	0	0
TATAAAATTTTTTTTTTAAA	20	2	2	0	1	0	1	0	0
TATAAAATTTTTTTTTTAA	19	3	2	0	1	0	2	0	0
TATAAAATTTTTTTTTTATTTTTTTATTTTTATTTAAATTTCCATTTAAT	50	4	1	0	0	0	4	0	0
TATAAAATTTTTTTTTA	18	7	2	0	1	0	6	0	0
TATAAAATTTTTTTTTTCTT	20	2	1	0	0	0	2	0	0
TATAAAATTTTTTTTTTCT	19	3	2	0	1	0	2	0	0
TATAAAATTTTTTTTTTC	18	5	2	0	3	0	2	0	0
TATAAAATTTTTTTTTTTCAAAGTTTC	27	2	2	0	1	0	1	0	0
TATAAAATTTTTTTTTTC	19	3	2	0	1	0	2	0	0
TATAAAATTTTTTTTTTTTC	20	2	1	0	0	0	2	0	0
TATAAAATTTTTTTTTTTTTCA	22	2	1	0	0	0	2	0	0
TATAAAATTTTTTTTTTTTTTATGT	25	2	1	0	0	0	2	0	0
TATAAAATTTTTTTTTTTTTTTA	23	2	1	0	0	0	2	0	0
TATAAAATTTTTTTTTTTTTTTC	23	2	1	0	0	0	2	0	0
TATAAAATTTTTTTTTTTTTTTTCA	26	2	1	0	0	0	2	0	0
TATAAAATTTTTTTTTTTTTTTTC	25	3	1	0	0	0	3	0	0
TATAAAATTTTTTTTTTTTTTTTTTC	27	2	1	0	0	0	2	0	0
TATAAAATTTTTTTTTTTTTTTTTTTC	28	2	1	0	0	0	2	0	0
TATAAAATTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT	42	2	1	0	0	0	2	0	0
TATAAAATTTTTTTTTTTTTTTTTTTTTTTTTTTT	34	3	1	0	0	0	3	0	0
TATAAAATTTTTTTTTTTTTTTTTTTTTTTTT	31	4	1	0	0	0	4	0	0
TATAAAATTTTTTTTTTTTTTTTTTTTTTT	30	6	1	0	0	0	6	0	0
TATAAAATTTTTTTTTTTTTTTTTTTTT	29	8	1	0	0	0	8	0	0
TATAAAATTTTTTTTTTTTTTTTTTTTT	28	9	2	1	0	0	8	0	0
TATAAAATTTTTTTTTTTTTTTTTTTT	27	11	2	1	0	0	10	0	0
TATAAAATTTTTTTTTTTTTTTTTTT	26	14	2	1	0	0	13	0	0
TATAAAATTTTTTTTTTTTTTTTTT	25	17	2	1	0	0	16	0	0
TATAAAATTTTTTTTTTTTTTTTT	24	20	2	1	0	0	19	0	0
TATAAAATTTTTTTTTTTTTTTT	23	23	2	1	0	0	22	0	0
TATAAAATTTTTTTTTTTTTTT	22	27	2	1	0	0	26	0	0
TATAAAATTTTTTTTTTTTTT	21	30	2	1	0	0	29	0	0
TATAAAATTTTTTTTTTTTT	20	33	2	1	0	0	32	0	0
TATAAAATTTTTTTTTTTT	19	36	2	1	0	0	35	0	0
TATAAAATTTTTTTTTTT	18	41	3	1	2	0	38	0	0
TATAAAATTTTTTTTTT	17	54	3	1	6	0	47	0	0
TATAAAATTTTTTTTT	16	79	3	2	13	0	64	0	0
TATAAAATTTTTTTT	15	116	3	8	18	0	90	0	0
TATAAAATTTTTTT	14	174	4	14	47	0	111	0	2
TATAAAATTTTTT	13	318	4	32	118	0	161	0	7
TATAAAATTTTT	12	701	4	93	299	0	294	0	15
TATAAAATTTT	11	1987	5	575	743	2	634	0	33
TATAAAATTT	10	4983	5	1588	1987	5	1319	0	84
TATAAAATT	9	12381	5	4103	4312	32	3716	0	218
TATAAAAT	8	37180	5	12568	10011	83	13735	0	783
TATAAAA	7	114137	6	39226	31168	369	40564	3	2807

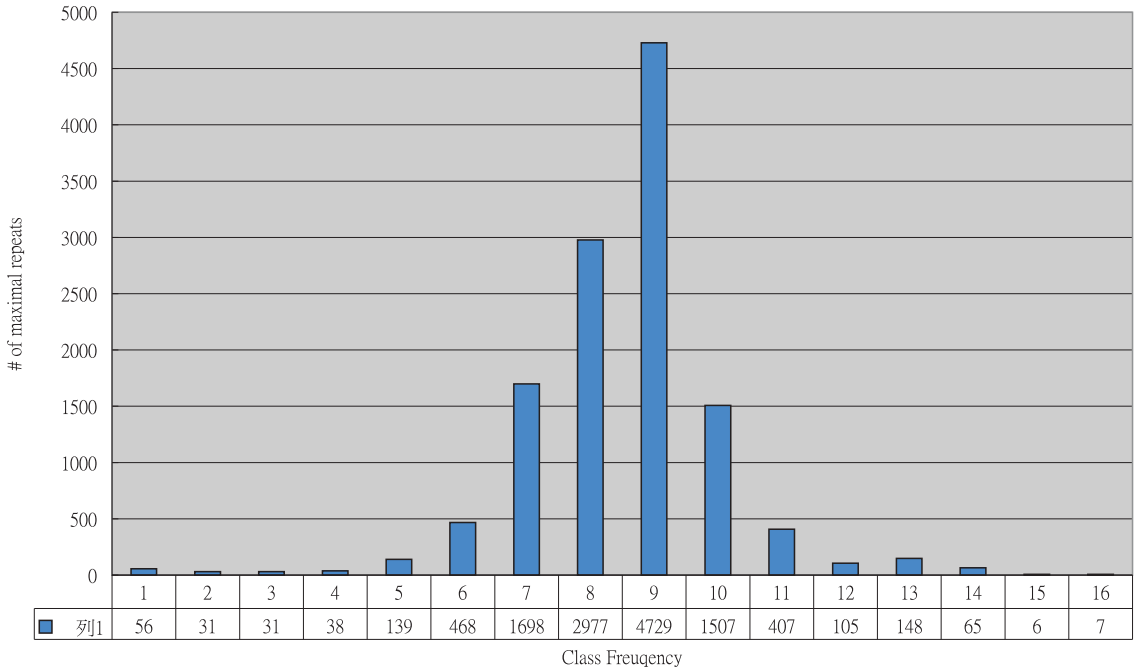


Figure 6: The class frequency of maximal repeats of yeast with $tf \geq 10$ and $len \geq 30$ (k=3,l=50)

to under 5 in this study because the default maximal number of opening files using Linux was $4^5 = 1024$ if we didn't recompiler the kernel.

4.2 The Computation Time via PC-clusters

The computation time of our approach could be reduced via parallel computing with PC-clusters by partitioning the computation into different PCs according to the prefix of the suffixes of DNA sequences. As shown in Fig 5 and Fig 6, we experimented with the DNA sequences of Yeast and used 8 PCs (8*CPU=P4 2.4G, RAM = 8*512MB and Red Hat Linux 9.0) to evaluate the speedup of the computation using parallel computing. The value of speedup we achieved was 4.94 when $k = 4$ and $l = 50$ using the Yeast's genome. Note that the speedup value when $k = 1$ was empty because the number of groups (files) was only $4^k = 4$. To focus on the approach of using external memory in this paper, we did not discuss the details about how to do parallel processing here. However, we will discuss that works on another paper in the future.

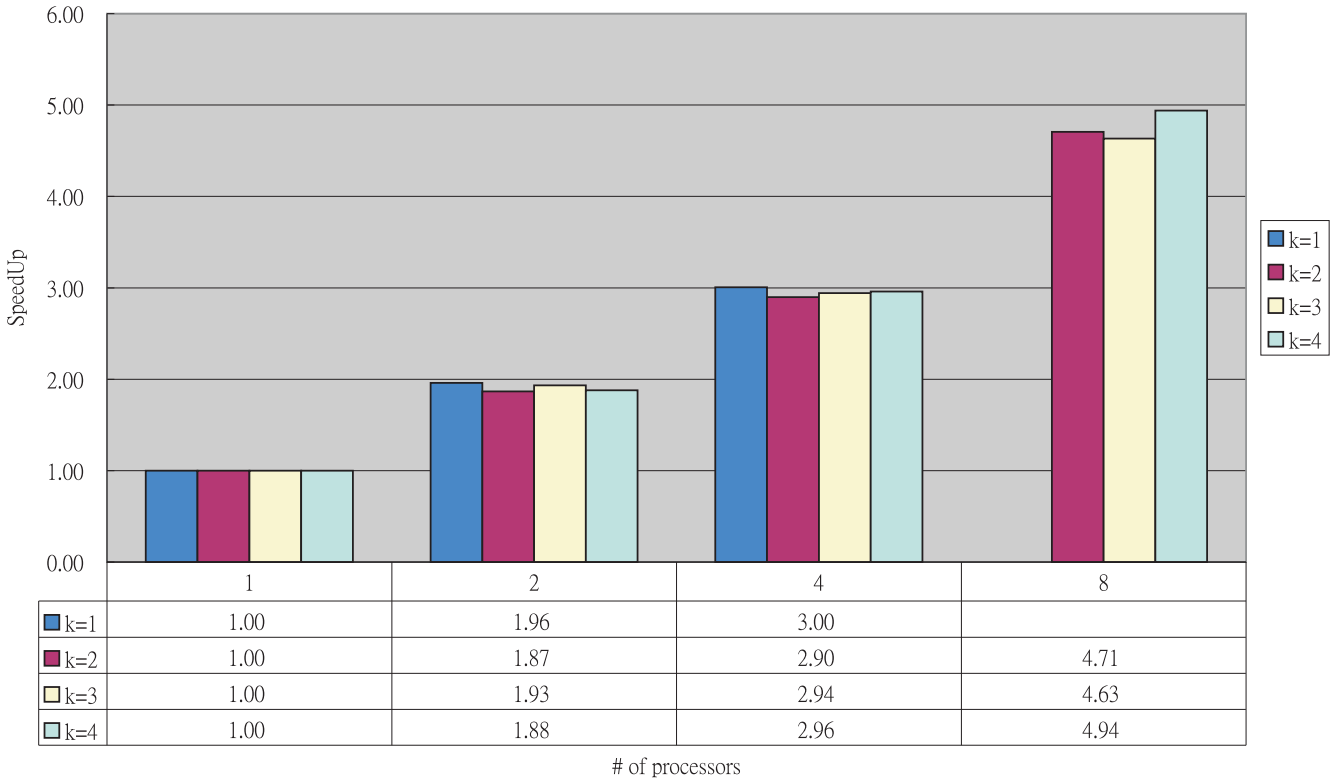
Acknowledgement

This work was partially supported by National Science Council, Taiwan, R.O.C., under grant NSC 93-2213-E-468-003. Thank the center of Bioinformatics of THMU to setup the computing environment. Thank Prof.Ng Ka-Lok for gathering the genomic sequences and thank Prof.Chang Pei-Jun for providing

Table 5: The total computation time(in seconds) with different number of PCs(Yeast,$l = 50$)

	The number of PCs			
	1	2	4	8
k=1	41105	20960	13679	
k=2	40968	21938	14140	8705
k=3	39740	20552	13504	8577
k=4	41716	22203	14093	8447

Table 6: The speedup with different number of PCs(Yeast,$l = 50$)



	1	2	4	8
k=1	1.00	1.96	3.00	
k=2	1.00	1.87	2.90	4.71
k=3	1.00	1.93	2.94	4.63
k=4	1.00	1.88	2.96	4.94

valuable suggestions. Finally, we thank the referees for giving the valuable suggestions to improve this paper.

References

[1] Mohamed Ibrahim Abouelhoda, Stefan Kurtz, and Enno Ohlebusch. The enhanced suffix array and its applications to genome analysis. In *Proceedings of the Second International Workshop on Algorithms in Bioinformatics*, pages 449–463. Springer-Verlag, 2002.

[2] V. Chizhikov, M. Wagner, A. Ivshina, Y. Hoshino, A. Z. Kapikian, and K. Chumakov. Detection and Genotyping of Human Group A Rotaviruses by Oligonucleotide Microarray Hybridization. *J. Clin. Microbiol.*, 40(7):2398–2407, 2002.

[3] Jeong-Hyeon Choi and Hwan-Gue Cho. Analysis of common k-mers for whole genome sequences using SSB-tree. *Genome Informatics*, 13:30–41, 2002.

[4] Thoman H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introdution to Algorithms, 2nd Edition*. MIT Press and McGraw-Hill, 2002.

[5] Jitender S. Deogun, Fangrui Ma, and Jingyi Yang. A prototype for multiple whole genome alignment. In *Proceedings of the 36th Hawaii International Conference on System Sciences(HICSS'03)*, 2003.

[6] Robert C. Edgar and Eugene W. Myers. PILER: identification and classification of genomic repeats. *Bioinformatics*, 21(suppl_1):i152–158, 2005.

[7] David Wang et al. Microarray-based detection and genotyping of viral pathogens. *Microbiology*, 99(24):15687–15692, 2002.

[8] Paolo Ferragina and Roberto Grossi. The string B-tree: a new data structure for string search in external memory and its applications. *Journal of the ACM*, 46(2):236–280, 1999.

[9] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences : computer science and computational biology*. Cambridge University Press, 1997.

[10] Samuel Karlin, Allan M. Campbell, and Jan Mrazek. Comparative dna

analysis across diverse genomes. *Annual Review of Genetics*, 32(1):185–225, 1998.

[11] Stefan Kurtz. Reducing the space requirement of suffix trees. *Software-Practice and Experience*, 29(13):1149–1171, 1999.

[12] Stefan Kurtz, Jomuna V. Choudhuri, Enno Ohlebusch, Chris Schleiermacherand Jens Stoye, and Robert Giegerich. REPuter: the manifold applications of repeat analysis on a genomic scale. *Nucleic Acids Research*, 29(22):4633–4642, 2001.

[13] A. Lefebvre, T. Lecroq, H. Dauchel, and J. Alexandre. FORRepeats: detects repeats on entire chromosomes and between genomes. *Bioinformatics*, 19(3):319–326, 2003.

[14] Webb Miller. Comparison of genomic DNA sequences: solved and unsolved problems. *Bioinformatics*, 17(5):391–397, 2001.

[15] Tom M. Mitchell. *Machine Learning*. The McGraw-Hill Companies, Inc, 1997.

[16] Krisztián Monostori, Arkady Zaslavsky, and Heinz Schmidt. Suffix vector: Space- and time-efficient alternative to suffix trees. In Michael J. Oudshoorn, editor, *Twenty-Fifth Australasian Computer Science Conference (ACSC2002)*, Melbourne, Australia, 2002. ACS.

[17] Snehasis Mukhopadhyay, Changhong Tang, Jeffery Huang, and Mathew Palakal. Genetic sequence classification and its application to cross-species homology detection. *The Journal of VLSI Signal Processing : Systems for Signal, Image, and Video Technology*, 35:273–285, 2003.

[18] James Noble and Charles Weir. *Small Memory Software: Patterns for systems with limited memory*. Addison Wesley Professional, 2001.

[19] Valerio Parisi, Valeria De Fonzo, and Filippo Aluffi-Pentini. STRING: finding tandem repeats in DNA sequences. *Bioinformatics*, 19(4):1733–1738, 2003.

[20] Alkes L. Price, Neil C. Jones, and Pavel A. Pevzner. De novo identification of repeat families in

large genomes. *Bioinformatics*, 21(suppl_1):i351–358, 2005.

[21] Baeza-Yates Ricardo and Ribeiro-Neto Berthier. *Modern Information Retrieval*. Addison Wesley, 1999.

[22] K. Rocha, C. Medeiros, M. Monteiro, L. Gonçalves, and P. Marinho. Design of specie-specific primers for virus diagnosis in plants with pcr. In *BIBE*, pages 149–158, 2004.

[23] Deepak Sharma, Biju Issac, G. P. S. Raghava, and R. Ramaswamy. Spectral Repeat Finder (SRF): identification of repetitive sequences using Fourier transformation. *Bioinformatics*, 20(9):1405–1412, 2004.

[24] Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne. *Applied Operating System Concepts*. John Wiley & Sons, 1 edition, 2000.

[25] Subbaya Subramanian, Vamsi M. Madgula, Ranjan George, Rakesh K. Mishra, Madhusudhan W. Pandit, Chanderashekar S. Kumar, and Lalji Singh. Triplet repeats in human genome:distribution and their association with genes and other genomic regions. *Bioinformatics*, 19(5):549–552, 2003.

[26] Li-Tong Tsay. A study on sorting string suffixes of large-scaled text collection in external memory. Master's thesis, Department of Computer Science and Information Engineering, National Chung Cheng University, Chiayi, Taiwan 62107, R.O.C., 2001.

[27] Jing-Doo Wang. A case study of predicting the class of the unknown viruses of ssRNA positive-strand in NCBI. In *The 21st Workshop on Combinatorial Mathematics and Computational Theory*, pages 249–256, 2004.

[28] Jing-Doo Wang and Jyh-Jong Tsay. Minging periodic events from restorpective Chinese news. *International Journal of Computer Processing of Oriental Languages, A Special Issue on "Web and WAP Oriental Languages Multimedia Computing"*., 15(4):361–377, 2002.

Appendix A

Some steps in candidate significant term extraction are shown below. For example, as shown in Fig. 7(A), The "A" whose length is 1 is a candidate significant term

and has the value of "cnt" as "2" when scanning to the suffix with the value of position as 7. When the next suffix, with the value of position as 1, is scanned, as shown in Fig. 7(B), the longest common prefix of two adjacent suffixes is obtained as "ACTT", whose length is 4 and exceeds the length 1 in the field, "len", at the top record of the stack. Then, the new record with "ACTT" is pushed into the stack. Note that the terms, "AC" and "ACT", are not pushed according to the definition of maximal repeat. When the next suffix is scanned, as shown in Fig. 7(C), the longest common prefix of two adjacent suffixes is shown as, "A", whose length is 1 and is shorter than that of 4, at the top record of the stack. Therefore, the record with the term, "ACTT", is popped and the statistics, "cnt=2", of that term are output. Because the term, "A", is a substring of, "ACTT". That is, the statistics concerning "ACTT" must be included in those concerning the term, "A". Using the similar processes as above, we could compute the other kinds of statistics, e.g. frequency distribution among classes, if each suffix tagged with specific label, e.g. class identity.

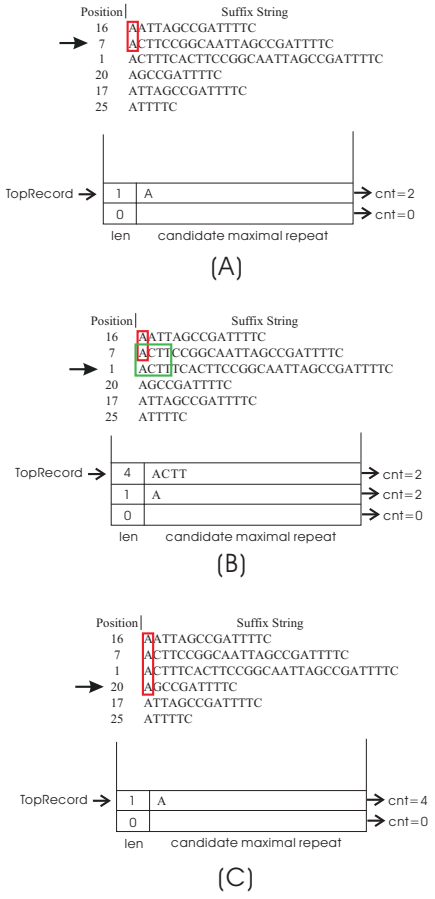


Figure 7: Extract candidate maximal repeats