Ladderlike Stepping and Interval Jumping Searching Algorithms for DNA Sequences

Tun-Wen Pai[1], Margaret Dah-Tsyr Chang[2], Jia-Han Chu[1], Wei-Yuan Chang[1], Hsiu Ling Tai[2]

[1]Department of Computer Science, National Taiwan Ocean University,[2] Institute of Molecular and Cellular Biology & Department of Life Science, National Tsin Hua University, Taiwan, R.O.C.

twp@mail.ntou.edu.tw dtchang@life.nthu.edu.tw

Abstract

In this study we have designed a novel algorithm for searching common segments in multiple DNA sequences. To improve efficiency in pattern searching, combination of hashing encoding, quick sorting and ladderlike stepping and/or interval jumping techniques are applied. Since multiple sequence alignment of DNA sequences from the giant genomic database is usually time consuming, we develop a three-phase methodology to search common sub-segments and reduce its time complexity for pattern matching. In the first coding phase, DNA nucleotide sequences are transformed into a numerical space set. Subsequently, the quick sort algorithms are employed in the second sorting stage to reorder the encoded data. In the last searching phase, ladderlike stepping and interval jumping rules are proposed to increase efficiencies of numerical comparison. In addition, two interval segmentation techniques, uniform partition and bitwise partition are applied prior to interval jumping procedures. The segmenting methodologies are designed according to the length of searching pattern, and the proposed ladderlike searching algorithms provide robust and improved performance. Experimental results show that the algorithms are capable of reducing time complexity from $O(mL_i(L_i-m+1)+mL_j(L_j-m+1))$ to $O(|I^i|+|I^j|)$.

Keywords: Stepping, Interval Jumping, Hashing Encoding

1 Introduction

In the post-genomic era, genomic DNA sequences of many species have been completely determined. Important subsequent studies are gene annotation and functional analysis (Stein, L. 2001), in order to gain insights in features, structures, and functions of the regulatory segments of genes (Pennacchio, L. 2001, Majewski, J. 2002). Generally, high similarities are found among nucleotide sequences and the deduced amino acid sequences of conserved protein families. Employing tools provided by existing DNA or protein databases, matching of such characteristics can be found quickly (GenomeNet 2003, JST 2003). However, vast areas on chromosomes still contain segments with unknown genetic functions, which require new bioinformatic tools to proceed analysis (Suzuki, Y. 2001, Davuluri, R. 2001, Loots G. 2002).

Chromosomes often contain special short DNA segments for protein recognition and binding. Such DNA and protein interaction can affect gene expression or protein functions. For example, the upstream promoter region of an eukaryotic gene possesses so called "transcription element" for regulatory protein to bind, and its corresponding protein is named as transcription factor (Pilpel, Y. 2001, Majewski, J. 2002, Trinklein, N. 2003). Though features of known short DNA sequences can be identified by several programs (Baldi, P. 2000, Buhler, J. 2001, Ning, Z. 2001, Volfovsky, N. 2001), they are often obtained by one-to-multiple string matching, which means sending a DNA string sequence into a database and retrieving segments with known features. However unknown or novel segments are difficult to be identified using the conventional methods. Thus, we have developed a complete program as a new solution and we can search for special consensus DNA sequences not well aligned in a set of gene family efficiently by employing the so called interval jumping methodology.

Time complexity required in the matching process is often used for evaluating the efficiency of a searching algorithm. Assume a certain DNA sequence with length n and a searching pattern with length m, the time complexity is $O(mn)$ while searching the sample in the sequence by traditional string matching algorithm. Assume the pattern with length m is the combination of all possible subsequence in a DNA sequence with length n, there are $n-m+1$ possible patterns requiring to be matched, and the time complexity increases dramatically to $O(mn\times(n-m+1))$. If there are K sequences with same length being matched with traditional matching algorithms, the time complexity is $O(mn\times(n-m+1)\times K)$ for finding common subsequence with length m in K sequences, which is extremely time consuming. Thus in the knowledge domain of pattern matching, except for traditional matching algorithm, numerous algorithms have been proposed for reducing time complexity, such as:Karp-Rabin algorithm (KR-A) (Karp, R.M. 1987, Crochemore, M. 1996), Knuth–Morris- Pratt algorithm (KMP-A) (Knuth, D.E. 1977, Knuth, D.E. 1998), Boyer-Moore algorithm (BM-A) (Boyer, R.S. 1977, Cole, R. 1994), quick search algorithm (QS-A) (Sunday, D.M. 1990), etc. Karp-Rabin algorithm uses hash functions to encode strings into digital numbers, and its time complexity is $O(m)$ in pre-processing encoding phase. For KR-A, the worst case in searching phase is $O(mn)$ and the average expected time complexity becomes $O(m+n)$. Measuring two sequences with same string length of

average expected time complexity is $O((n-m+1)\times(m+n))$. Deriving this in the same fashion, if *K* sequences with same length are proceeding common segment matching, $O((n-m+1)\times(m+n)\times K)$ is average expected time complexity. Detailed comparisons are shown in Table 1, where PPTC means Pre-processing Phase Time Complexity, SPTC means Searching Phase Time Complexity, ECTC means Expected Case Time Complexity or Best Case Time Complexity. Worst case time complexity are shown in the SPTC column, where *m* refers to pattern length, *n* refers to sequence length, and σ refers to the size of the bad character table in BM algorithm. The KR algorithm proposed by Karp and Rabin encodes and transforms strings into digital number sets, and then it proceeds number comparison to reduce required string matching time. Knuth-Morris-Pratt algorithm and Boyer-Moore algorithm provided the concept of interval matching methodology. Should we integrate the features of these algorithms, the time complexity of searching repeating DNA sequences can be reduced.

	PPTC	SPTC	ECTC (BCTC)
BF-A	-	O(mn)	O(mn)
KR-A	O(m)	O(mn)	O(m+n)
KMP-A	O(m)	O(n+m)	-
BM-A	O(m+σ)	O(mn)	O(n / m)

Table 1：Comparison between different algorithms

2 Definitions

The main purpose of this study is to search for common segments with variable length among DNA family sequences. A DNA sequence is a table of string composed of $\Sigma = \{A, C, G, T\}$, where *A*, *C*, *G*, *T* are the four base pairs in a DNA sequence. Let *S* represents the set of DNA family sequences, y represents a DNA sequence in the *S* set, $y \in S$, where $|y|$ is the length of sequence *y*. The index of sequence y is from 0 to $|y|-1$, and the subsequence of the index from i to j is represented as $y[i...j]$, where $0 \le i \le j \le |y|-1$. If set ψ^y is the set of all possible subsequences from sequence *y*, where each subsequence with different length is one candidate matching pattern, ψ_m^y is the subset of ψ^y containing patterns of length *m*. If w represents a certain pattern, $|w|$ represents its length, and its index is from 0 to $|w|-1$. Suppose there are in total N sequences in the *S* set, where S^i represents the i^{th} sequence with length $|S^i| = L_i$.

The searching algorithms discussed in this paper, strings are first transformed into a new numerical set, represented as $\tilde{\psi}^y$, and $\tilde{\psi}_m^y$ is the subset of $\tilde{\psi}^y$ with pattern length m. Then numbers in set $\tilde{\psi}_m^y$ are sorted and grouped according to their values, so that numbers within common range could gather in a same interval set I, where I^y represents the encoded interval set for sequence y, and $|I^y|$ is the quantity of interval for the encoded sequence y. ${}^A I^y$ represents the position of a encoded number A in encoded intervals for sequence y, it also represents the interval index of A.

3 Algorithm Description

In this study we proposed two algorithms for matching common substrings in multiple DNA sequences. The first algorithm is Ladderlike Stepping Searching Algorithm (LSSA), the second algorithm is Ladderlike Interval Jumping Searching Algorithm (LIJSA). Both algorithms contain three phases: encoding, sorting, and searching phase. Both the first and second phases possess same procedures in proposed algorithms, while in the last searching phase, the Ladderlike Interval Jumping Searching Algorithm plays a role as a more advanced module and faster matching performance. The key feature in encoding phase is that while string sequences are being transformed into numerical sequences, the encoded numbers are with unicity, which can reduce the time complexity comparing with the conventional one-by-one string matching. The main purpose in sorting phase is to match sequentially by the value of numbers to avoid redundant and unsorted sequence searching. The last searching phase proceeds and records the final matching process. LSSA requires no additional interval segmentation procedure, while LIJSA collects encoded numbers in the same interval after sorting. LIJSA performs interval segmentation in different ways in order to extend the dissimilarity among encoded numbers in the same interval. The modification can greatly increase the probability of interval jumping, and decrease the cost of matching procedures. The detailed procedure and efficiency improvement of the three phases are discussed as follows:

3.1 Coding phase

The employed coding algorithms are similar to Karp-Rabin algorithms (Ricardo, B.Y. 1992). But in our proposed algorithms, the conversion radix is 4, and G,A,T,C are replaced with 0,1,2, and 3, instead of the ASCII encoded numbers. The benefit of using 4 as radix is the "unicity" characteristic while transforming string pattern into numbers, hence the coding conflict in Karp-Rabin algorithm which take coding radix less than ASCII number can be avoided. We obtain the "unique" hash number with pattern length *m* in each segmented subsequence by applying equation (1) and (2), while *w* is pattern string with length *m*, *base* is the coding radix, and *y* is the sequence for coding phase. The time complexity in the hashing process is $O(N \times (L_{\max} + m - 1))$, while *N* is the number of family sequence set, $L_{\max}$ is length of the longest sequence in the set, *m* is the length of a pattern, where $2 \le m \le L_{\max}$.

$$hashfunc(w[0..m-1]) =$$

$$w[0]base^{m-1} + w[1]base^{m-2} + ... + w[m-1]base^{0} \quad (1)$$

$$rehashfunc \quad (y[i], y[i+m], hash([i...i+m-1])) \quad (2)$$

3.2 Sorting Phase

From the first character of a DNA sequence with length $L_{\max}$ to its end, we sequentially obtain *m* base pairs by fixed length and proceed hashing encoding, which will extract $L_{\max} - m + 1$ patterns and generate its corresponding numerical values. Then Quick Sort algorithms will sort the encoded values in numerical order and the time complexity of Quick Sort is $O(n \log n)$, where $n = L_{\max} - m + 1$ (Hoare, C.A.R. 1962, Knuth, D.E. 1998).

3.3 Searching Phase

Both proposed Ladderlike Stepping Searching Algorithm (LSSA) and Ladderlike Interval Jumping Searching Algorithm (LIJSA) suggest to compare two encoded and ordered sequences at a time, and then use the results to compare with the rest sequences in a family set progressively. In both proposed algorithms, two numerical sets derived from two sequences are being matched, and the matching process is like walking down from a ladder. The former LSSA algorithm can treat each encoded value as a single step or interval without any interval segmentation process, thus its matching is single stepping. The latter LIJSA algorithm generates interval grouping and segmentation in order to perform single stepping jumping or multiple intervals jumping during matching processes. The comparison between these two algorithms is in the next section. In the beginning of matching phase, our algorithm picks up any two sequences for matching. While searching for common subsequence with length *m*, after comparing the two encoded numerical set, same values will form a new set $\tilde{\psi}_m^{i,j}$ and those elements can be decoded as the common strings of the two comparing sequences, represented as $\psi_m^{i,j}$. If *m>1*, elements in set $\psi_m^{i,j}$ are consequently less than the shorter encoded DNA sequence, i.e. $|\psi^{i,j}| \le \min(L_i, L_j)$, $1 \le i, j \le N$ and $i \ne j$. The new encoded numeric set $\tilde{\psi}_m^{i,j}$ is then compared with the other encoded numeric sequence $\tilde{\psi}_m^{k}$ by the same method. As the number of matching ladder decreases while comparison goes forward, the probability of finding equal value also decreases, that is, common strings are less likely found in the latter comparisons. As a result, the matching time complexity must decrease as the comparison goes forward. In multiple DNA family sequence sets, the best case and worst case are as following: the best case occurs while, searching for common subsequences by a pattern with length *m*, and in the two sequences with no common encoded value, that is, $|\psi_m^{i,j}| = 0$, so time complexity becomes only $O(L_i + L_j)$. The worst case occurs while, after each matching common encoded value is all the encoded value of the shorter sequence, that is, $|\psi_m^{i,j}| = \min(L_i, L_j)$, and the time complexity is $O(L_i + L_j + \min(L_i, L_j) + L_k + \min(\min(L_i, L_j), L_k) + L_m + ...)$. As for cases when each sequence is the subsequence of other sequences, $S^i \subset S^j \subset S^k$, the time complexity is approximately $O(N \times \min(L_1, L_2, L_3, ...))$. Even in the worst case, the time complexity is still lower than traditional searching algorithms. Except for finding common segments among all the designated sequences quickly, this algorithm can also obtain the location and number of occurrence of a common segment among sequences. Searching results can be provided to biologists for timely further analysis of comparing common DNA sequences and factors of special function in the same DNA family.

3.4 Advanced Stepping Searching Algorithms

The main point in the searching phase is proceeding ladderlike searching on two encoded sequences. But for further improvements on the time complexity, we propose a more advanced method, hoping for gathering values within a similar range so that interval jumping could be applied. By jumping over elements smaller than original encoded value, the pattern can be compared directly with value intervals equal or larger than it. The more intervals skipped, the faster the searching phase can be finished. Like going down from a ladder, the matching can be performed step by step, sometimes it can jump over several ladders, thus the algorithm is called Ladderlike Interval Jumping Algorithm. There are two interval segmentation methods to define the actual range of a ladder in the LIJSA. The first partition is called bitwise interval segmentation, and the second one is named uniform interval segmentation. The advantage of bitwise interval segmentation is its ability to rapidly calculate the interval location that encoded value belongs to. Bitwise interval segmentation is realized according to the searching pattern length (*m*), its formula is as following:

$$|I_m| = \lceil \log_{base} 4^m \rceil \quad (3)$$

The *base* value is set to 10. From equation (3), we find the quantity of intervals is an integer number equal to the ceiling of $log_{base}4^m$. The first interval is the single bit interval, second is two-bit interval, and third is three-bit interval, and so on. The meaning of bit here is the length of a certain value, for example, value of 0~9 are in the first interval, 10~99 are in the second interval, and so on. Because bitwise interval segmentation can be done in encoding phase, it does not require any further computing during searching phase. In the preceding searching phase, stepping rules are still as illustrated, but jump rule (JR) is applied before matching two sequences. JR is described as the following,

$$\begin{aligned}
&case1: {}^{A}I_m^1 \ne {}^{B}I_m^2 \quad and \quad A > B \quad \Rightarrow \quad B \quad jump to \quad {}^{A}I_m^2 \\
&case2: {}^{A}I_m^1 \ne {}^{B}I_m^2 \quad and \quad A < B \quad \Rightarrow \quad A \quad jump to \quad {}^{B}I_m^1 \\
&case3: {}^{A}I_m^1 = {}^{B}I_m^2 \quad and \quad A \ne B \quad \Rightarrow \quad step by step in \ LSSA \\
&case4: {}^{A}I_m^1 = {}^{B}I_m^2 \quad and \quad A = B \quad \Rightarrow \quad check if \ A and B \ repeat \\
&\qquad\qquad\qquad\qquad\qquad\qquad continuously \ in \ interval I_m^1 \ and I_m^2
\end{aligned} \quad (4)$$

where ${}^{A}I_{m}^{1}$ represents the interval position of number A in the first encoded sequence. ${}^{B}I_{m}^{2}$ is the second interval position of number B in the second encoded sequence. If the two numbers being matched are not in the same interval, the sequence in the interval with smaller numbers must jump to the interval with larger numbers, and then continues with stepping searching in LSSA. If the two numbers being matched are in the same interval, directly employ the stepping searching algorithms in LSSA. If the segment contents are different, their relatively encoded values would be different. And if they are distributed in different intervals, apparently, jumping rules can be applied for interval jumping, and lower down the time complexity in searching phase. After running bitwise interval segmentation, theoretical analysis and experimental results in Figure 1 shows the number of values in all intervals. Most values of encoded strings fall into the last two intervals in segmentation region, thus the last two segmentation regions should be emphasized. Applications on uniform interval segmentation method are proposed in this study. Only by finding rules of interval segmentation and the amount of segmented intervals, then we can decide the grouping method of all encoded values.

There are two types for segmentation region in this algorithm. The first is general segmentation region and the second is uniform interval segmentation region. General segmentation regions are treated as non-concentrated intervals, which are the sets other than the last two intervals. I_{m_G} represent the non-concentrated intervals with patterns of length *m*, they require no interval segmentation because of their low proportion. The amount of intervals of general segmentation region is $|I_{m\_G}|=1$. Most encoded values take place in uniform interval segmentation regions. In this algorithm, before partitioning the concentrated intervals, the gap length between intervals is first analyzed statistically by histogram equalization, the results are shown in Figure 2. After segmentation, values in every interval are distributed uniformly, thus the probability of interval jumping is increased.

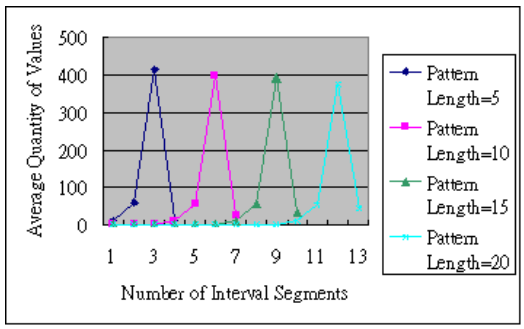

Figure 1: After bitwise segmentation on RNase family sequence, average interval quantity are distributed mostly in the last two intervals.

Original amount of intervals is $\lceil \log_{base} 4^{m} \rceil$, where *base* is 10, and is represented by $|I_{m\_original}|$. If without uniform interval segmentation, the algorithm directly calculates the segmentation base, named $sbase = 10^{|I_{m\_original}|-2}$. For example, given a string of length 5, and the original amount of intervals is $\lceil \log_{10} 4^{5} \rceil = 4$, $sbase = 10^{4-2} = 100$, the amount of last second interval $|I_{m\_L2}|=9$, and the amount of last interval is $|I_{m\_L1}| = (4^{m} - 10^{|I_{m\_original}|-2}) / (10^{|I_{m\_original}|-1})$ and equal to 1, so the total amount of intervals is $|I_{m\_total}| = |I_{m\_G}| + |I_{m\_L1}| + |I_{m\_L2}| = 11$. There will be 11 segmented intervals for pattern length equal to 5.

The uniform interval segmentation method is also accomplished in the encoding phase, so there is no additional time complexity required. We can also obtain the location of the interval where value A belongs to by formula $\lfloor A/sbase \rfloor$, then sequence searching procedures are done by applying JR, same as bitwise segmentation method discussed before. If there are no elements matched in the corresponding intervals, the pointer sequentially jumps to the next interval until it meets an interval containing elements. The ultimate purpose of either bit interval segmentation or uniform interval segmentation is try to lower down the time complexity in the algorithm, hoping we can avoid matching every single encoded data value. Furthermore, by the interval jumping method, we can speed up searching and lower the expected time complexity from $O(L_i + L_j)$ to $O(|I^i| + |I^j|)$.

4 Results And Discussions

In the previous discussion of ladderlike searching algorithm for common segments from a specific DNA family, stepping searching algorithm results in a better performance than traditional brute-force and Karp-Rabin algorithms. Tables 2 shows complete comparison among these developed algorithms. From the row of LSSA, the time complexity for encoding, sorting, and searching phases are $O((L_{\max} - m)^2 \times N)$, $O(N \times (L_{\max} - m) \log (L_{\max} - m))$, and $O(N \times (L_{\max} - m))$ respectively. The ratio between LSSA and KR-A is able to be represented by $\log(L_{\max} - m) / L_{\max}$, and the value is less than 1. Furthermore, the ratio between KR-A and BF-A is $O((L_{\max} - m)/(L_{\max} m))$, and is less than 1 as well. Therefore, searching complexity for LSSA is obviously less than KR-A and BF-A. The paper also provides two advance techniques by segmenting encoded data set to improve the performance of LSSA. Both bitwise and uniform interval segmentation provide expected time complexity $O(|I^i| + |I^j|)$, which is less than LSSA in general cases. In order to compare the details and performance of both interval segmentation algorithms, here provides the discussion of variance of intervals from encoded data set. While the variance increased by interval segmenting procedures, the performance of time complexity is decreased as we expected. From Figure 3, we can observe that both bitwise and uniform interval segmentation possess much higher variability than original encoded sequence, and the relation between variance and pattern length are shown as well. When the length of searching pattern increased, the relative variances in bitwise interval segmentation decreases. This is due to the

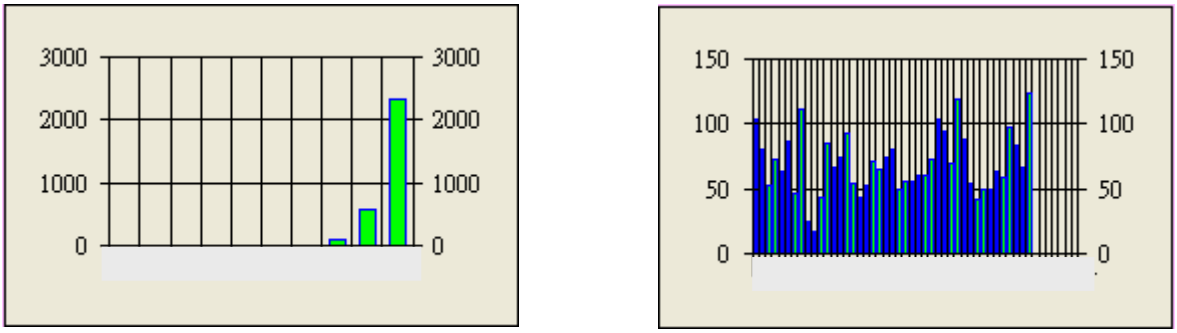

Figure 2 RNase sequence distribution before segmentation(Left) and after segmentation (Right)

reasons of longer searching pattern length transformed into larger encoded digital numbers, and most of encoded data will be confined in the last few intervals. Hence, the variance of interval becomes decreasing as pattern length increasing. However, the variability of uniform interval segmentation possesses better stability when searching pattern length increased. Since the range of interval is still limited in a certain number, no matter how the length of searching pattern changes, the encoded data quantity in an interval still maintain a higher variance than LSSA. Through above discussion, the proposed algorithms suggest to employ bitwise interval segmentation technique with respect to a shorter searching pattern length, and employ uniform interval segmentation technique for a longer searching pattern length. Both advance partition technique will enhance the interval jumping possibility in terms of searching algorithms.

To employ the proposed novel algorithms on DNA sequence sets, we have initially analyzed a set of mammalian promoter sequences retrieved from the NCBI database. We focus on searching consensus sequence motifs with length from 8 to 20 as the conventional transcription elements. Three consensus motifs containing 20 nucleotides, one each with 8, 10, 14, and 19 nucleotides could be identified from the alpha-fetoprotein promoters of *Gorilla gorilla* (AB053570), *Pan troglodytes* (AB053571), *Mus musculus* (AB053573), and human (AB053572). Similarly, 16 consensus motifs of 8 to 16 nucleotides were identified from the growth hormone promoters of human (K00470), *Pan troglodytes* (AF374233), black-handed spider monkey (AF374235), rhesus monkey (U02293), bovine (M57764), and procine (M17704). In addition, 13 consensus motifs of 8 to 10 nucleotides were identified from the RHAG promoters of *Gorilla gorilla* (AF177628), *Macaca mulatta* (AF177631), *Pan troglodytes* (AF177627), *Pongo pygmaeus* (AF177629), *Papio hamadryas* (AF177632), *Hylobates sp*. (AF177630), *Mus musculus* (AB036994), and human (AF178744).

In comparison to the results using the conventional transcription element searching program GenomeNet, the known transcription elements such as CdxA(WWTWMTR), could be identified from all three types of promoters by both methods. However, some of the CdxA were not well aligned in the promoter sets. In addition, an SRY (AAACWAM) not aligned in the RHAG promoters, an USF (NCACGTGN) and two cap (NCANNNNN) were identified in the growth factor promoters by our program. The results revealed that some known transcription elements located differently in the promoters upstream of the same gene family. Furthermore, we have found some un-characterized conserved sequence motifs which may or may not be aligned by conventional multiple sequence alignment, indicating that they could be potential novel transcription elements should their transcription activity be confirmed with further promoter activity assays.

We have demonstrated that our algorithm performed efficiently and provided new information in the multiple sequence comparison effectively.

Acknowledgement:

This research was supported partly by NSC91-3112-B-007-004, NSC92-3112-B-007-001, and MOE Program for Promoting Academic Excellence of Universities under the grant number 89-B-FA04-1-4.

4 References

Stein, L. (2001):Genome annotation : from sequence to biology. *Nature reviews genetics*, **2**:493 – 503.

Majewski, J. and Ott, J. (2002):Distribution and characterization of regulatory elements in the human genome, *Genome research*, **12**:1827~1836.

Pennacchio, L. and Rubin, E.(2001):Genomic strategies to identify mammalian regulatory sequences, *Nature reviews genetics*, **2**:100~109.

GenomeNet: Human Genome Center (HGC) of the University of Tokyo, http://www.genome.ad.jp/ , Accessed 14 Aug. 2003.

JST: Software and Tools for Genome Analysis, http://www-btls.jst.go.jp , Accessed 14 Aug. 2003.

Suzuki, Y., Tsunoda, T., Sese, J., Taira, H. *et.al*(2001): Identification and characterization of the potential promoter regions of 1031 kinds of human genes, *Genome research*, **11**:677–684.

Loots, G., Ovcharenko, I., Pachter, L., Dubchak, I., and Rubin, E.(2002):rVista for comparative sequence-based discovery of functional transcription factor binding sites, *Genome Research*, **12** : 832 – 839.

Davuluri, R., Grosse, I., Zhang, M.(2001): Computational identification of promoters and first exons in the human genome, *Nature Genetics,* **29**: 412 – 417.

Trinklein, N., Force Aldred, S., Saldanha, A.,and Myers, R.(2003): Identification and functional analysis of human transcriptional promoters, *Genome research*, **13**:308~312.

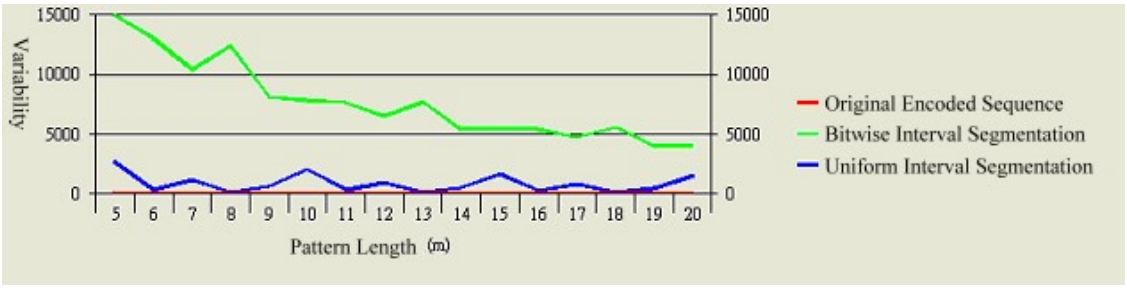

Figure 3 : Variance of Original Encoded Sequence, Bitwise and Uniform Interval Segmentation

Majewski, J. and Ott, J.(2002):Distribution and characterization of regulatory elements in the human genome, *Genome research*, **12**:1827~1836.

Pilpel, Y., Sudarsanam, P., Church, G.(2001): Identifying regulatory networks by combinatorial analysis of promoter elements, *Nature Genetics* ,**29**:153~159.

Buhler, J.(2001):Effective large-scale sequence comparison by locality-sensitive hashing. *Bioinformatics*, **17**:419-428.

Ning, Z.,Cox AJ, Mullikin JC.(2001): SSAHA:a fast search method for large DNA databases, *Genome Research*, **11**:1725-1729.

Volfovsky, N., Haas, B.J. and Salberg S.L.(2001):A clustering method for repeat analysis in DNA sequences, *Genome Biology*.

Baldi, P. and Baisnee, P. (2000): Sequence analysis by additive scales : DNA structure for sequences and repeats of all lengths, *Bioinformatics*, **16**: 865-889.

Karp, R.M.**,** Rabin, M.O.(1987):Efficient randomized

pattern-matching algorithms. *IBM J. Res. Dev.*, **31**(2):249-260.

Crochemore, M., Lecroq, T.(1996): Pattern matching and text compression algorithms, in *CRC Computer Science and Engineering Handbook*, A. Tucker ed., Ch. 8, 162-202, CRC Press Inc., Boca Raton, FL.

Knuth, D.E.**,** Morris**,** J.H.**,** Pratt, V.R.(1977): Fast pattern matching in strings, *SIAM Journal on Computing*, **6**(1):323-350.

Knuth, D.E.(1998), vol. 3 of *The Art of Computer Programming,* 2nd ed., Addison Wesley.

Boyer, R.S.**,** Moore, J.S.(1977):A fast string searching algorithm. *Communications of the ACM.* , **20**:762-772.

Cole, R.(1994):Tight bounds on the complexity of the Boyer-Moore pattern matching algorithm, *SIAM Journal on Computing*, **23**(5):1075-1091.

Sunday, D.M.(1990):A very fast substring search algorithm, Communications of the ACM., **33**(8):132-142.

Ricardo, B.Y., Gonnet G.H.(1992):A new approach to text searching, *Communications of the ACM*, **35**(10): 74-82.

Hoare, C.A.R. (1962): Quicksort, *The Computer Journal*, **5**:10-15.

	Time Complexity		
BF-A	$O((L_{\max} - m) \times L_{\max} \times m \times N)$		
	Encoding	Sorting	Searching
KR-A	$O((L_{\max} - m)^2 \times N)$	-	$O((L_{\max} - m)^2 \times N)$
LSSA (Original Encoded Sequence)	$O((L_{\max} - m)^2 \times N)$	$O(N \times (L_{\max} - m) \log(L_{\max} - m))$	$O(N \times (L_{\max} - m))$
LIJSA (Bitwise Interval Segmentation)	$O((L_{\max} - m)^2 \times N)$	$O(N \times (L_{\max} - m) \log(L_{\max} - m))$	$O(N \times (L_{\max} - m))$ $O(N \times (L_{\max} - m) / \lvert I_{\max} \rvert)$
LIJSA (Uniform Interval Segmentation)	$O((L_{\max} - m)^2 \times N)$	$O(N \times (L_{\max} - m) \log(L_{\max} - m))$	$O(N \times (L_{\max} - m))$ $O(N \times (L_{\max} - m) / \lvert I_{\max} \rvert)$
Note:	m: pattern length, N : number of sequences, $L_{\max}$:length of the longest sequence in family, $\lvert I_{m\_\max} \rvert$: number of interval segment of the longest sequence in family		

Table 2 : Time Complexity Comparison for BF-A, KR-A, LSSA, and LIJSA