Algorithms for Molecular Biology BioMed Central

Research

Open Access

Finding the region of pseudo-periodic tandem repeats in biological sequences

Xiaowen Liu* and Lusheng Wang*

Address: Department of Computer Science, City University of Hong Kong, Kowloon, Hong Kong

Email: Xiaowen Liu* - liuxw@cs.cityu.edu.hk; Lusheng Wang* - lwang@cs.cityu.edu.hk

* Corresponding authors

Published: 28 February 2006

*Algorithms for Molecular Biology*2006, **1**:2 doi:10.1186/1748-7188-1-2

Received: 23 February 2006

Accepted: 28 February 2006

This article is available from: http://www.almob.org/content/1/1/2

Abstract

Summary: The genomes of many species are dominated by short sequences repeated consecutively. It is estimated that over 10% of the human genome consists of tandemly repeated sequences. Finding repeated regions in long sequences is important in sequence analysis.

We develop a software, LocRepeat, that finds regions of pseudo-periodic repeats in a long sequence. We use the definition of Li et al. [1] for the pseudo-periodic partition of a region and extend the algorithm that can select the repeated region from a given long sequence and give the pseudo-periodic partition of the region.

Availability: LocRepeat is available at http://www.cs.cityu.edu.hk/~lwang/software/LocRepeat

Background

Finding pseudo-periodic repeats (or tandem repeats) is an important task in biological sequence analysis [1-3]. The genomes of many species are dominated by short sequences repeated consecutively. It is estimated that over 10% of the human genome consists of tandemly repeated sequences. About 10–25% of all known proteins have some form of repeated structure ranging from simple homopolymers to multiple duplications of entire globular domains. An instance (originally from Jaitly et al. [2]) of a human tandem repeat appears below (Genbank:10120313):

CCTCCTCCTCCACCTCCTCCTCCTCCTCCTCCTCC-CGCCTTCTCATCCTCCTCCACTT

CCTCCTCCTCCTCCTCCTCCCCTTCTCATCCTCCTC-CTCTTCATCTACCC

This tandem repeat consists of 35 approximate copies of the repeated pattern CCT.

Variation in the pseudo-periodic repeats demonstrates biologically important information. Sensitive tools for finding those regions containing pseudo-periodic repeats are required in practice. Repeats occur frequently in biological sequences, but they may not be exact in many cases. If the repeats are exact, the problem can be easily solved from computation point of view. However, repeats are seldom exact in biological sequences. The errors in those repeats make it difficult to find regions of those repeats. Many measures and algorithms have been proposed.

Landau and Schmidt [4] studied the problem of finding the two consecutive copies in a sequence of length n such that the edit distance (a match costs 0 and a mismatch/indel costs 1) between the two copies is at most k. The running time of the algorithm is $O(kn \log k \log(n/kL))$.

Table 1: Pseudo periodic repeats of 1SRY (matrix:blosum62, gap penalty: -4)

Unit	Pseudo-periodic unit	Length	Similarity with previous unit
1	MVDLKRLR	8	
2	QEPEVFHR	8	-5
3	AIREKGVA	8	-10
4	LDLEALLA	8	-1
5	LDREVQEL	8	7
6	KKRLQEVQ	8	6
7	TERNQVA	7	6
8	KRVPKAP	7	-4
9	PEEKEAL	7	-2
10	IARGKAL	7	3
11	GEEAKRL	7	3
12	EEALRE	6	10
13	KEARLE	6	12
14	ALLLQV	6	-6
15	PLPP	4	-8

Schmidt [5] used weighted grid digraphs for finding all non-overlapping pairs of substrings (not necessarily consecutive) with the highest scores in a given string of length n. The algorithm can handle any score scheme. It requires $O(n^2 \log n)$ time and $\Theta(n^2)$ space. In both [4] and [5], only two copies of the pattern are considered.

Measures for finding repeats

Three measures can be used to give partitions of repeated regions.

Quasiperiodicty

Wan and Song proposed a measure in which all the repeated copies (except the last one) have the same length [6]. For this measure, a linear time and space algorithm was given [6].

Approximate periods

Sim et al. [7] introduced a notion of approximation periods (*approximate period*) using edit distance or relative edit distance. The problem in general is defined as follows: given a string x, find a repeated pattern p such that x can be partitioned as $x = p_1 p_2 \ldots p_k$ and $\max_{l=1}^{k} d(p, p_l)$ is minimized. Here $d(p, p_l)$ is the relative edit distance which is $\frac{1}{L} \times$ the edit distance, where $L = (|p| + |p_l|)/2$ is the average length of the two strings p and p_l. Note that, the normalization of the edit distance is important for finding repeated patterns since otherwise, one can give a partition in which each pattern has one letter and the edit distance is at most 1 (small). The problem in general is NP-hard [7]. When the repeated pattern p is assumed to be a substring of x, The problem can be solved in $O(|x|^4)$ time. Note that the second measure is more general than the first since it allows insertions and deletions. Both measures in [7] and [6] use the bottleneck function that finds the repeated pattern p and assumes that each copy p_i in the long string is close to the repeated pattern p, i.e., $d(p_i, p) \leq \delta$ and δ is minimized. However, in biological sequences, copies of the repeated patterns may change gradually so that some repeats in the region may have very little in

Table 2: Pseudo periodic repeats of LPXA_ECOLI (matrix:blosum62, gap penalty: -4)

Unit	Pseudo-periodic unit	Length	Similarity with previous unit
1	MIDKSAFVHPTAIVEEGA	18	
2	SIGANAHIGPFCIVGPHV	18	14
3	EIGEGTVLKSHVVVNGHT	18	16
4	KIGRDNEIYQFASI	14	-7
5	GEVNQDLKYAGEPTR	15	-3
6	VEIGDRNRIRESVTI	15	2
7	HRGTVQGGGL	10	-14
8	TKVGSDNLLMINAHIAHD	18	-24
9	CTVGNRCILANNATLAGH	18	20
10	VSVDDFAIIGGMTAVHQF	18	4
11	CIIGAHVMVGGCSGV	15	4

```
    aaa aat att ttt tta
aaa aat att ttt tta
```

Figure 1
The alignment for $\pi(s_A)$.

common. For example, it is well-known that the N-terminal non-globular region of *Thermus thermophilus* seryl-tRNA synthetase (PDB:1SRY) [1,8] has weak 7-residue repeats. See Table 1. The similarity score between two consecutive patterns is calculated using Blosum62 matrix and the gap penalty is set to be -4. The repeated patterns gradually changes from the 4-th unit LDLEALLA to the 13-th unit KEARLE. The average similarity score for the nine pairs of consecutive patterns is 4.56. But the similarity score between the 4-th unit and the 8-th unit is -11. In this case, the algorithms based on the bottleneck function may fail to find the multiple repeats.

Pseudo-periodic repeats

Li et al. [1] gave the first measure that allows gradual changes of patterns and changes of pattern lengths in the region. The repeats they defined are called the *pseudo-periodic* repeats. Given a repeated region (a string) x and a partition $X = s_1 s_2 ... s_k$, the *pseudo periodic score* is

$$\sum_{i=1}^{k-1} d(s_i, s_{i+1}) + c \times (|s_1| + |s_k|),$$

where $d(\cdot)$ is the edit distance, $|s_i|$ is the length of s_i, and c is a factor that control the penalty of the two ends of the partition. Li et al. [1] gave a $O(|x|^2)$ algorithm to compute an optimal partition of a given repeated region x. It was shown that the pseudo-periodic score can accurately give partitions for tandem repeated regions, where the repeated patterns are weakly similar.

Example: The example is from [1]. The sequence of the LbH domain of members of the LpxA family consists of the imperfect tandem repetition of hexapeptide units [9-11]. These imperfect tandem repeats (partitions) have been accurately detected by the algorithm using the pseudo periodic score [1]. (See Table 2).

In sequence analysis, we may have a long sequence s and only a substring t (or a few substrings) of s contains the consecutive repeats. The problem here is to find out the substring t and give an optimal pseudo-periodic partition. We call this problem the *local pseudo-periodic* problem. In this paper, we define the maximization version of the pseudo-periodic partition and develop an algorithm that solves the local pseudo-periodic problem in $O(n^2)$ time, where n is the length of the input sequence s.

Definitions

In this section, we first give a definition of the *pseudo-periodic partition* of a string that is originally proposed in Li et al. [1]. We then give a definition of the *local pseudo-periodic partition* of a string.

Pseudo-periodic Partition

Let $s = a_1 a_2 ... a_n$ be a string of length n. A *partition* $\pi(s) = \{s_1, s_2, ..., s_k\}$ of s is a set of substrings of s such that $s = s_1 s_2 ... s_k$ (s_i's are also called *repeats*). When s is clear, we use π instead of $\pi(s)$. $\Pi(s)$ denotes the set of all partitions of s. Let s_i and s_{i+1} be two strings. The *similarity measure* $\mu(s_i, s_{i+1})$ between s_i and s_{i+1} is the maximum alignment value for s_i and s_{i+1}. For any two letters (possibly spaces) x and y, $\mu(x, y)$ is the similarity score between the two letters. For example, one can use the following score scheme I: a match costs 1, a mismatch costs -1, and an insertion or deletion costs -1. Here we choose to use maximization version since for protein sequences, there are popular similarity matrices, e.g., PAM matrix.

```
     aaaa aa-at aaa
aaaa aaat aaa
       (a)

     aaaa aaat aaa
aaaa aaat aaa- -
       (b)

     aaaa aaat aaa
aaaa aaat aaa
       (c)
```

Figure 2
The alignment for $\pi(s_B)$.

Table 3: Results for the speed test of LocRepeat

Length	2000	4000	6000	8000	10000
DNA	0.5s	2.2s	4.8s	7.0s	10.8s
PROTEIN	1.1s	4.3s	10.0s	17.7s	26.9s

Let c be a negative constant. We call $\frac{c}{2}$ the *granularity factor*. Let Δ denotes a space in an alignment. In this paper, we assume that $\frac{c}{2} > \mu(x, \Delta)$ for any letter x in the given sequence.

Let us consider the following example. $s_A = s_1s_2s_3s_4s_5$ and $\pi(s_A) = \{s_1, s_2, s_3, s_4, s_5\}$, where $s_1 = aaa$, $s_2 = aat$, $s_3 = att$, $s_4 = ttt$ and $s_5 = tta$. The self-alignment of this partition is show in Figure 1. The value of the self-alignment is $\sum_{i=1}^{4}\mu(s_i, s_{i+1}) + \frac{c}{2}(|s_1| + |s_5|)$, where $\frac{c}{2}|s_1|$ and $\frac{c}{2}|s_5|$ are the penalty scores for the two segments s_1 and s_5 aligned to spaces.

Note that the score for insertion and deletion would be different from the granularity factor $\frac{c}{2}$. If there is a gap at the right end of the alignment between s_4 and s_5, there is ambiguity in the calculation of the self-alignment value. Therefore, we need a more precise definition for the value of the self-alignment corresponding to a partition.

i / j	6	5	4	3	2	1	0
C 1						C -2	-∞
A 2					C -4	A -2	-∞
G 3				C -6	A -4	G -2	-∞
A 4			C -8	A -6	G 6	A -2	-∞
G 5		C -10	A -8	G -6	A 16	G 7	-∞
T 6	C -12	A -10	G -8	A 5	G 6	T -2	-∞

A G A G
A G A G

Figure 3
Dynamic programming algorithm and local alignment for s = CAGAGT.

Let $s = s_1s_2...s_k$ be the string and $\pi(s) = \{s_1, s_2, ..., s_k\}$. $|s|$ denotes the length of the string. $pre(s, i)$ is the length-i prefix of s and $suf(s, i)$ is the length-i suffix of s. Note that the gap at the right end of the self-alignment of s only appears in the segments s_{k-1} and s_k. Denote by s_e the suffix $s_{k-1}s_k$ of s. We can designate that only the last i letters in s_e are mapped to spaces with score $\frac{c}{2}$ each, for $1 \le i \le |s_e|$. Now let us consider the remain part $pre(s_e, |s_e| - i)$ of s_e. There are two cases: (1) If $i \ge |s_k|$, $pre(s_e, |s_e| - i)$ is a prefix of s_{k-1} and is optimally aligned with s_k. (2) If $i < |s_k|$, $pre(s_e, |s_e| - i)$ contains s_{k-1} and a prefix of s_k. In this case, s_{k-1} is optimally aligned with s_k and the letters in the prefix of s_k are scored as $\mu(x, \Delta)$ each.

For a partition π of s and a fixed i, $1 \le i \le |s_e|$, let $V(\pi, c, i)$ be the value of the self-alignment such that s_1 is mapped to spaces with score $\frac{c}{2}$ each, s_j is optimally aligned with s_{j+1} for $j = 1, 2, ..., k-2$, $pre(s_e, |s_e| - i)$ is scored according to the above two cases, and the last i letters in s_e are mapped to spaces with score $\frac{c}{2}$ each. We have

$$V(\pi,c,i) = \begin{cases} \sum_{j=1}^{k-2}\mu(s_j, s_{j+1}) + \mu(pre(s_e, |s_e| - i), s_k) + \frac{c}{2}\times(|s_1| + i) & \text{if } i \ge |s_k|; \\ \sum_{j=1}^{k-1}\mu(s_j, s_{j+1}) + \mu(x, \Delta)\times(|s_k| - i) + \frac{c}{2}\times(|s_1| + i) & \text{if } i < |s_k|. \end{cases}$$

In $V(\pi, c, i)$, the alignment between $s_1s_2...s_{k-2}pre(s_e, |s_e| - i)$ and $s_2s_3...s_k$ is called the *middle* alignment. The value of the self-alignment of π is defined as $V(\pi, c) = \max_{i=1}^{|s_e|} V(\pi, c, i)$.

For example, let $s_B = s_1s_2s_3$ and $\pi(s_B) = \{s_1, s_2, s_3\}$, where $s_1 = aaaa$, $s_2 = aaat$ and $s_3 = aaa$. We use score scheme I and $c = -1$. The valid value of i is 1, 2, ..., 7 since $|s_2s_3| = 7$. For i

Table 4: Local optimal pseudo-periodic region for PRNP

Unit	Position	Length	Unit
1	215–238	24	ggtggtggctgggggcagcctcat
2	239–262	24	ggtggtggctgggggcagcctcat
3	263–286	24	ggtggtggctgggggcagccccat
4	287–310	24	ggtggtggctggggacagcctcat
5	311–327	17	ggtggtggctggggtca

= 5 ≥ $|s_3|$, $pre(s_2, 2)$ is optimally aligned with s_3, s_1 and $suf(s_2s_3, 5)$ is scored as $\frac{c}{2}$ (Figure 2(a)). So $V(\pi(s_B), c, 5) = \mu(s_1, s_2) + \mu(pre(s_2, 2), s_3) + \frac{c}{2} \times (|s_1| + 5) = -\frac{3}{2}$. For $i = 2 < |s_3|$, s_2 is optimally aligned with s_3, $pre(s_3, 1)$ is scored as $\mu(x, \Delta)$ and $suf(s_3, 2)$ is scored as $\frac{c}{2}$ (Figure 2(b)). In this case, $V(\pi(s_B), c, 2) = \mu(s_1, s_2) + \mu(s_2, s_3) + \mu(x, \Delta) \times 1 + \frac{c}{2} \times (|s_1| + 2) = 0$. For $i = 4$, at the right ends of the optimal self-alignment of $\pi(s_B)$ (Figure 2(c)), there are 4 letters that match spaces. The last letter t in s_2 matches a space at the right end of the alignment. The assumption that $\frac{c}{2} > \mu(x, \Delta)$ forces this column to have score $\frac{c}{2}$ instead of $\mu(t, \Delta)$ to maximize $V(\pi, c)$. We have $V(\pi(s_B), c) = V(\pi(s_B), c, 4) = 1$. For $i = 1, 3, 6, 7$, the values are lower than $V(\pi(s_B), c) = V(\pi(s_B), c, 4) = 1$.

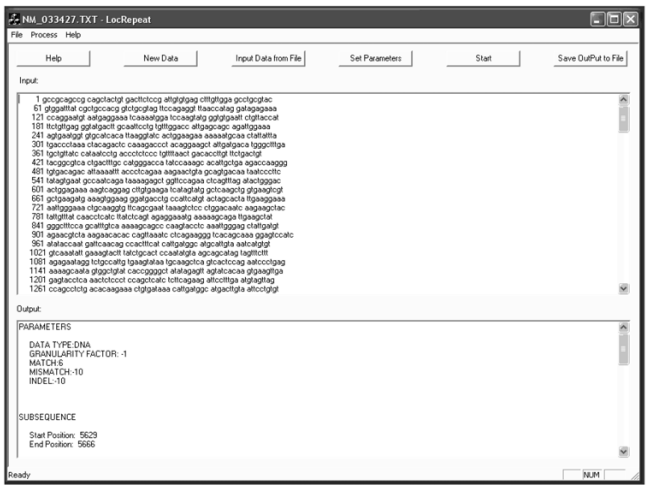

Figure 4
LocRepeat interface.

Let $\Pi(s)$ be the set of all possible partitions of s. $B_c(s) = \max_{\pi \in \Pi(s)} V(\pi, c)$ is the optimal $V(\cdot)$ value of partitions. A partition $\pi_q = \{s_1, s_2, ..., s_k\}$ of s is called the pseudo-periodic partition of s if $B_c(s) = V(\pi_q, c)$. In Li et al. [1], it was demonstrated that the numerical measure $B_c(s)$ (in fact, the minimization version) is sensitive for partitioning s into repeats that allow the gradual changes of patterns and changes of pattern lengths.

In practice, we are given a long string s. We want to find a region (substring) t of s that contains pseudo-periodic repeats. Once the region t is found, we want to get the pseudo-periodic partition of t. The mathematical problem is defined as follows:

Local pseudo-periodic partition problem

Given a string s, find a substring t (the local optimal pseudo-periodic region) of s such that

$$B_c(t) = \max_{t' \in Sub(s)} B_c(t'),$$

where $Sub(s)$ is the subset of all substrings of s.

The algorithm

Let s be the given string. We want to find a substring t of s with the maximum self-alignment value . Let $s[1, j]$ be the substring of s that consists of the first j letters. Informally, we use $w(i, j)$ to denote the maximum self-alignment value of a suffix t_j of $s[1, j]$ such that there are i letters at the right end of the self-alignment of t_j that are aligned with spaces and scored as $\frac{c}{2}$. Note that, the right end of the self-alignment of t_j could contain more than i spaces. However, only the last i spaces are scores as $\frac{c}{2}$ each and the rest of them are scored as the score for $\mu(x, \Delta)$.

Let T_j be the set of all suffixes of $s[1, j]$. For a substring t of s and an integer i, $\Pi(t, i) = \{\pi(t) | \pi(t) \in \Pi(t)$ and $|t_{k-1}t_k| \geq i\}$, where t_{k-1}, t_k are the last two repeats in $\pi(t)$. We define

$$w(i, j) = \max_{t \in T_j \& \pi(t) \in \Pi(t,i)} V(\pi(t), c, i) \qquad (1)$$

Table 5: Local optimal pseudo-periodic region for LGR6

Unit	Position	Length	Unit
1	30–53	24	LSMNNLTELQPGLFHHLRFLEELR
2	54–77	24	LSGNHLSHIPGQAFSGLYSLKILM
3	78–101	24	LQNNQLGGIPAEALWELPSLQSL D
4	102–124	23	LNYNKLQEFPVAIRTLGRLQELG
5	125–148	24	FHNNNIKAIPEKAFMGNPLLQTI H
6	149–172	24	FYDNPIQFVGRSAFQYLPKLHTLS
7	173–195	23	LNGAMDIQEFPDLKGTTSLEILT
8	196–219	24	LTRAGIRLLPSGMCQQLPRLRVLE
9	220–241	22	LSHNQIEELPSLHRCQKLEEIG
10	242–265	24	LQHNRIWEIGADTFSQLSSLQAL D
11	266–289	24	LSWNAIRSIHPEAFSTLHSLVKLD
12	290–311	22	LTDNQLTTLPLAGLGGLMHLKL

to be the maximum $V(\cdot, c, i)$ value of all the partitions in $\Pi(t, i)$, where t is a substring in T_j. To compute $w(i, j)$ using dynamic programming method, we first consider the boundary values of $w(i, j)$. We set $w(0, j) = -\infty$ since we do not allow $suf(t_{k-1}t_k, i)$ to be empty. Note that, by definition, $i \le j$.

Lemma 1 *For a sequence s of length n,* $w(j, j) = c \cdot j$ for $1 \le j \le n$.

Proof. For a partition $\pi(t) = \{t_1, t_2, ..., t_k\}$ satisfying $t \in T_j$ and $\pi(t) \in \Pi(t, j)$, from the definition of $\Pi(t, j)$, $|t_{k-1}t_k| \ge j$. Since t is a suffix of $s[1, j]$ and $|t| \ge |t_{k-1}t_k| \ge j$, we have $t = s[1, j]$ and $1 \le k \le 2$. Consider the self alignment of $\pi(t)$ such that the last j letters in t are mapped to spaces with score $\frac{c}{2}$. Two cases arise. Case 1: $k = 1$ and $t = t_1 = s[1, j]$. In this case, the middle alignment is empty. Thus, $V(\pi(t), c, j) = \frac{c}{2} \times (|t_1| + j) = c \cdot j$. Case 2: $k = 2$ and $t = t_1t_2 = s[1, j]$. In this case, the middle alignment is the alignment between $|t_2|$ spaces and t_2. By the assumption that $\frac{c}{2} > \mu(\Delta, x)$, $V(\pi(t), c, j) = |t_2| \times \mu(\Delta, x) + \frac{c}{2} \times (|t_1| + j) < c \cdot j$. □

From the above analysis, the initial and boundary values are

$$w(0, j) = -\infty,\ w(j, j) = c \cdot j \qquad (2)$$

Theorem 2 *For a sequence s of length n and* $2 \le j \le n$, $1 \le i < j$,

$$w(i, j) = \max \begin{cases} c \cdot i, \\ w(i, j-1) + \mu(s[j-i], s[j]), \\ w(i-1, j-1) + \mu(\Delta, s[j]) + \frac{c}{2}, \\ w(i+1, j) + \mu(s[j-i], \Delta) - \frac{c}{2}. \end{cases} \qquad (3)$$

Proof. Consider the partition $\pi(t) = \{t_1, t_2, ..., t_k\}$ such that $t \in T_j$, $\pi(t) \in \Pi(t, i)$ and $V(\pi(t), c, i) = w(i, j)$. We analyze the value of $V(\pi(t), c, i)$ based on different cases.

case 1. $\pi(t)$ has only one repeat. We have $|t| = |t_1| = i$ and the middle alignment is empty. Therefore $V(\pi(t), c, i) = \frac{c}{2} \times (|t_1| + i) = c \cdot i$.

case 2. $\pi(t)$ has $k \ge 2$ repeats. In this case, the middle alignment is not empty since it contains t_k. The last column in the middle alignment has three configurations: (a) the last column contains two letters $s[j - i]$ and $s[j]$, (b) the last column contains a space and the letter $s[j]$, (c) the last column contains the letter $s[j - i]$ and a space. For sub-case (a), if we take away the last letter $s[j]$ from the self alignment of $\pi(t)$, we can get a self alignment of $\pi'(t')$, where $t' \in T_{j-1}$, $\pi'(t') \in \Pi(t', i)$ and $V(\pi'(t'), c, i) = w(i, j - 1)$. By comparing the two self alignment, we have $V(\pi(t , c, i = V(\pi'(t'), c, i) + \mu(s[j - i], s[j]) = w(i, j - 1) + \mu(s[j - i], s[j])$. For sub-case (b), if we take away the last letter $s[j]$ and space aligned with $s[j]$ from the self alignment of $\pi(t)$, we can get a self alignment of $\pi'(t')$, where $t' \in T_{j-1}$, $\pi'(t') \in \Pi(t', i - 1)$ and $V(\pi'(t'), c, i) = w(i - 1, j - 1)$. Notice that in

the end of the self alignment of $\pi'(t')$, there are only i - 1 letters mapped to spaces with score $\frac{c}{2}$, and there is one more letter in the self alignment of $\pi(t)$ mapped to spaces with score $\frac{c}{2}$. Thus, $V(\pi(t), c, i) = w(i - 1, j - 1) + \mu(\Delta, s[j]) + \frac{c}{2}$. For sub-case (c), $\pi(t) \in \Pi(t, i + 1)$. We can impose that the alignment of the letter $s[j - i]$ and the space is scored as $\frac{c}{2}$, not $\mu(s[j - i], \Delta)$. From $V(\pi(t), c, i + 1) = w(i + 1, j)$, we have $V(\pi(t), c, i) = w(i + 1, j) + \mu(s[j - i], \Delta) - \frac{c}{2}$. □

Based on Theorem 2, a dynamic programming algorithm is designed. Let n be the length of the input sequence s. We compute $w(i, j)$ in the order shown below:

for j = 1 **to** n **do**

  for i = j **downto** 1 **do**

    compute $w(i, j)$ based on Theorem 2.

Obviously, the time complexity is $O(n^2)$, where n is the length of the whole string. A standard backtracking process allows us to find the local optimal pseudo-periodic region t.

The following example illustrates the algorithm. Let s = *CAGAGT*. We set c = -2 and use the following score scheme: a match costs 10, a mismatch costs -10, and an insertion or a deletion costs -10. The table constructed by using the dynamic programming algorithm is shown in Figure 3. The table is constructed in from the top to the bottom. For every row in the table, the $w(i, j)$'s are computed from left to right. From the table, it is easy to see that the maximum value of $w(i, j)$ is $w(2, 5) = 16$. From the maximum value $w(2, 5) = 16$, we know that the local optimal pseudo-periodic region t is a suffix of $s[1, 5]$ = *CAGAG* and there are 2 letters aligned with spaces and scored as $\frac{c}{2}$ at the right end of the self alignment of the local optimal pseudo-periodic region t. From $w(2, 5)$, we can backtrack $w(2, 5) \rightarrow w(2, 4) \rightarrow w(2, 3)$ and stop at $w(2, 3)$ since $w(2,3)$ gets its value from $c \cdot i$ indicating the first segment of the partition of t ends at 3-th letter in s and the length of the segment is 2. Thus, we get t = *AGAG*. From the self alignment, it is easy to get the partition of $\pi(t)$ = {*AG*, *AG*}.

The space complexity required is also $O(n^2)$ if we are not careful. However, we can release the space whenever they are no longer useful. Thus, only two columns, are required for the computation. For each of the two columns, we use two arrays: one array stores the value of $w(i, j)$ and the other array stores the starting position of the subsequence t that maximizes $w(i, j)$. Therefore, the space complexity is $O(n)$ for computing all $w(i, j)$'s. After $w(i, j)$'s are computed, we know the substring t that leads to the optimal w value. Therefore, we can reconstruct the alignment for t in $O(n_1^2)$ time and space, where n_1 is the length of t, the repeated region. If n_1 is still big, we can use the standard technique in [12] to reduce the space to $O(n_1)$ by doubling the computation time for reconstructing the alignment of t.

In practice, a sequence may contain more than one repeated region. To find all the repeated regions, we can select the best k values of $w(i, j)$'s for some pre-defined value k. Each backtracking gives a repeated region. Another way to set a threshold for the value of $w(i, j)$ and select all $w(i, j)$'s with value greater than the threshold.

Implementation

We have implemented the algorithm using Visual C++ 6.0 and Windows XP. The software is called LocRepeat and has a user-friendly GUI (See Figure 4). Another version without GUI that works for Linux is also available.

LocRepeat accepts three kinds of sequence: DNA, RNA and Protein. The user can either click 'New Data' button to directly input the sequence at the input area, or click 'Input Data from File' button to input a sequence from a file. The user can click 'Set Parameters' button to set parameters, such as granularity factor, gap penalty and similarity score matrix. After the sequence is input and the parameters are set, click the 'Start' button to begin the computation.

Experiment results

We have done experiments to test the speed and sensitivity of the software.

Speed Testing

The time complexity of the algorithm is $O(n^2)$. To test the speed in practice, we use arbitrarily generated DNA and protein sequences. We ran our software on a PC with Pentium 4 3.4G CPU and 1GB memory, the result is shown in Table 3. We can see that for long DNA and protein

sequences, our software can get the result in short time. For example, if the length of the sequence is 10000, it takes about 10.8 seconds and 26.9 seconds for DNA sequences and protein sequences, respectively. In some real applications, the length of sequences could be much longer than 10000. In this case, one can cut the long sequence into several short pieces and find out the repeated regions for each piece. If a region covers two pieces, then we can re-cut that segment to get that region.

Sensitivity testing using real data

We applied LocRepeat to the DNA sequence gene PRNP which contains tandem repeats (GenBank:M13667). The length of the sequence is 2420. We find the local optimal pseudo-periodic region [215,327], that contains 5 pseudo-periodic units (Table 4). The pseudo-periodic region misses the first several sites of the tandem repeats, but the region and the partitions show the tandem repeats correctly.

We also applied LocRepeat to the protein sequence LGR6 (Swiss-Prot: Q9HBX8). The length of the sequence is 828. We use PAM120 as the similarity score matrix and find the local units (Table 5).

In conclusion, the algorithm presented in this paper offers the possibility to find regions of pseudo-periodic repeats in a long sequence.

Acknowledgements

We thank the referees for their helpful suggestions. This work is fully supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China [Project No. CityU 1070/02E].

References

1. Li L, Jin R, Kok P, Wan H: **Pseudo-periodic partitions of biological sequences.** *Bioinformatics* 2004, **20:**295-306.
2. Jaitly D, Kearney PE, Lin G, Ma B: **Methods for reconstructing the history of tandem repeats and their application to the human genome.** *Journal of Computer and System Sciences* 2002, **65(3):**494-507.
3. Tang M, Waterman M, Yooseph S: **Zinc finger gene clusters and tandem gene duplication.** In *Proceedings of the Fifth Annual International Conference on Computational Biology, April 22–25 2001* Montreal, Canada, ACM; 2001:297-304.
4. Landau GM, Schimidt JP: **An algorithm for approximate tandem repeats.** In *Proceedings of the Fourth Annual Symposium on Combinatorial Pattern Matching* New York, LNCS 684, Springer-Verlag; 1993:120-133.
5. Schmidt JP: **All highest scoring paths in weighted grid graphs and their application to finding all repeats in strings.** *SIAM Journal on Computing* 1998, **27:**972-992.
6. Wan H, Song E: **Quasiperiodic biosequences and modulo incidence matrices.** *Proceedings of the 16th International Parallel and Distributed Processing Symposium* 2002:280.
7. Sim JS, Iliopoulos CS, Park K, Smyth WF: **Approximate period of strings.** *Theoretical Computer Science* 2001, **262:**557-568.
8. Biou V, Yaremchuk A, Tykalo M, Cusack MS: **The 2.9 Å crystal structure of *T. thermophilus* seryl-tRNA synthetase complexed with tRNA(Ser).** *Science* 1994, **263:**1404-1410.
9. Vaara M: **Eight bacterial proteins, including UDP-*N*-acetylglucosamine acyltransferase (LpxA) and three other transferases of *Escherichia coli*, consist of a six-residue periodicity theme.** *FEMS Microbiology Letter* 1992, **76:**249-254.
10. Vuorio R, Harkonen T, Tolvanen M, Vaara M: **The novel hexapeptide motif found in the acyltransferases LpxA and LpxD of lipid A biosynthesis is conserved in various bacteria.** *FEBS Letter* 1994, **337:**289-292.
11. Raetz CRH, Roderick SL: **A left-handed parallel beta helix in the structure of UDP-*N* -acetylglucosamine acyltransferase.** *Science* 1995, **270:**997-1000.
12. Myers EW, Miller W: **Optimal alignments in linear space.** *Computer Applications in the Biosciences* 1988, **4:**11-17.