Pacific Symposium on Biocomputing 4:202-213 (1999)

# Time and Memory Efficient Algorithm for Extracting Palindromic and Repetitive Subsequences in Nucleic Acid Sequences

Tatsuhiko TSUNODA , Masao FUKAGAWA, and Toshihisa TAKAGI
*Genome DB, HGC, Institute of Medical Science, University of Tokyo, 4-6-1, Shirokanedai, Minatoku, Tokyo, 108-8639, JAPAN*

**KWs:** Palindrome, Repeat, Efficient Pattern Extraction, DNA structure, RNA structure.

Genomic science and structural biology meet in the relationship between the sequence and the structure of nucleic acids. The structure that supports each function is preserved in the process of evolution as specific sequences. Particularly, the same sequence which appears in a different place such as a palindromic or repetitive sequence has biophysical meaning: recognition site of dimers, forming stem-loops, and contributions to global structure of nucleic acids. Also, the genetic network, transduction pathway, and tissue specificity largely depend on these. Although the relationship between them can be found experimentally, there is increasing demand for automated analysis. Especially, it is desirable to extract the same character sequences of arbitrary length (especially, very long ones) which co-occur at an arbitrary separation. We propose an algorithm to identify the maximum match sequence at each position with a calculation cost of $O(N \log N)$ and memory space of $O(N)$. Applying it to some sequences, we found unexpectedly large palindromes and repeats in DNA.

## 1 Introduction

It is quite common that the same sequence of characters appears at different positions in nucleic acids. First of all, let's classify such cases: (1) Direct repeats: simple repetitive sequences. (2) Trans-strand repeats: reverse sequences on the complimentary strand. (3) Backward repeats (ss-palindrome): reverse sequences with the same polarity. Although they look like palindromes as strings, because nucleic acids have polarity, they are not palindromes. (4) Inverted repeats (ds-palindrome, so-called palindrome in nucleic acids): repetitive sequences on the complimentary strand of nucleic acids.

One of the simplest methods for generating such repetitive sequences is to use reverse-transcriptase. If mRNA transcripted from a subsequence of DNA is put in by the reverse-transcriptase in the same direction near the original site, it provides a direct repeat. If it is put in to the complimentary chain with the opposite direction, it provides a palindrome. There are many other biological mechanisms which produce repetitive and palindromic sequences. If there is no biological mechanism which uses such sequences, even if they are generated, they will be lost before long by rearrangement and mutation in the process of evolution.

Table 1: Phenomena based on palindromic and repetitive sequences.

| Influence on | Mechanism | Instance (phenomenon) |
|---|---|---|
| DNA | a. Site recognized by proteins (homo-dimers etc.) | Transcription factor binding sites, restriction enzyme recognizing sites, and prokaryotic operators. |
| | b. Stem-loop formation | Prokaryotic transcription terminators, prokaryotic transcription attenuators, and viral replicating origins. |
| | c. Duplicate, copy | High density sites, retro-transposons. |
| RNA | a. Complex (double helix + protein) | RNA duplex forming then protein binding sites (formation of functional protein-RNA complex). |
| | b. Stem-loop formation | RNA processing sites (self splicing) and translation regulators[1]. |
| | c. Structure modification | RNA enhancers (Tat RNA which binds *tar* RNA of HIV[4]). |
| Protein | Hydrophobicity | Self-folding hydrophobic structure of protein translated from palindrome[6]. |

However, recent research has clarified that they may be very important. One is the interaction between proteins and nucleic acids. For example, some transcription factors form homo-dimers and recognize DNA by their 3D structure. The recognition sites of homo-dimers that form in tandem are repetitive sequences. While, the recognition sites of homo-dimers that form symmetrically are palindromic sequence. Moreover, it is well known that in a palindrome of the nucleic acids, the subsequence binds with the subsequence of the opposite direction and complimentary bases on its own strand and tends to make a stem-loop. In addition, it is known experimentally that backward repeats and trans-strand repeats form higher-order structures depending on conditions, although its biophysical meaning is not clear yet.

The phenomena which are clarified at present are classified in Table 1. This table is a summary of the cases where palindromic and repetitive sequences interact locally. However, they also have the possibility to form a global 3D structure of the nucleic acids. In the case of repetitive sequences, after loosing the site, they bind with the sequences on a complimentary chain at other sites occasionally. In general, the probability of such cases is low. However, they sometimes occurs when the nucleic acids binds and it becomes very dense, forming a triple helix or quadruple helix nucleic acids. Also in the case of palindromes, there is the possibility to bind with another reverse-complimentary sequences at a distant position, which makes a higher-order 3D structure of the nucleic acids.

When these 3D structures are made in the transcription regulatory re-

gion, by controlling the distance between the transcription factors, they control whether each transcription factor can bind with the region, or determine whether they can cause the interaction. It affects the amount of transcription and the expression of the mRNA and protein. Splicing of RNA, translation regulation, and even the structure and function of proteins may depend on this mechanism. As already classified above, people have laboriously investigated experimentally the biological meaning of these unusual sequences such as palindromes. They examined especially palindromes without spacers. However, because the phenomena are known to differ according to the presence of spacers and the differences in length, it is necessary to examine palindromes or repeats with arbitrary length spacers. Also, to consider the global structure, it is necessary to search subsequence pairs with long distances on long sequences such as full genomes in the future.

When looking for palindromes, repeats, and other variations with fixed length and without spacer between each pair, the search can be processed at a time proportional to the sequence length $N$. It is because it only has to prepare a matching pattern in temporary memory of a fixed length for each subsequence in the full sequence, and match the pattern in the position with a fixed distance from each original subsequence. However, because we want to extract such pairs of subsequences of variable length with unknown spacer length, i.e. with arbitrary distance between them, naively, the calculation time is $O(N^2)$. Although we can consider converting the entire sequence into a suffix array or a tree structure beforehand and using a lot of memory ($4^L$, where $L$ is the upper limit of the length) to reduce the processing time to $O(N)$, it is not realistic considering the amount of the memory which can be used and the size of full genomes.

Thus, here we set the task to detect palindromes, complimentary chain repeats, and inverted repeats of maximum length at each position with arbitrary spacers by a complete match search on large size nucleic acid sequences. The reason we limited the task to complete match is that it is one of the most fundamental parts of matching; we can collect them to find more flexible matches. We propose a high-speed ($O(NlogN)$), memory space efficient ($O(N)$) algorithm that satisfies the above requirement.

## 2 Algorithm

### *2.1 Originality*

In general, repetitive subsequences on an input sequence are detected by the following procedures: First, subsequence (S) is extracted from the full sequence. Secondly, S is converted to target subsequence S'. (E.g., for searching

repetitive sequences, S itself. For searching palindromic sequences, a complementary sequence of S.) Lastly, the target subsequence S' is searched in the original full sequence. In this method, the process costs up to $O(N^2L^2)$, where $N$ is the length of the full sequence, and $L$ is the upper limit of the length of the subsequences (the searching template is extracted from each position of the full sequence. In addition, this process is applied to all range of $L$ to find maximum length subsequences at each position).

However, our method differs from this. It converts the entire sequence into another sequence, and compares it with the original sequence. For instance, to find a palindrome, a complementary sequence is made first and compared with the original sequence. As a result, all the problems we want to solve can be converted to the problem of a string comparison between two texts. In the comparison of two strings, a high speed algorithm with a small memory space can be developed using sorting. These are the original points of our algorithm.

### *2.2 Sequence conversion*

First of all, from the input sequence, the following four sequences are generated (Fig. 1):

(0) A capitalized sequence of the original sequence. Thus it can uniformly deal with the sequence without making distinction between capital letters and small letters in the original sequence.

(1) A complementary-base substituted sequence of (0). If we compare it with (0), we can detect complementary sequence repeats (trans-strand repeats) with reversed polarity.

(2) A reverse directed sequence of (0). If we compare it with (0), we can detect reverse direction repeats (backward repeats).

(3) A complementary sequence of (0). If we compare it with (0), we can detect palindromic subsequences (symmetrical repeats, so-called inverted repeats).

### *2.3 Algorithm of extracting subsequences common in two sequences*

(a) Substring sorting in each text (Fig. 2(left)) : First, scanning the sequence from the beginning to the end, subsequence (within length L, 256 characters in our implementation) which starts from each position is listed. Secondly, all the subsequences are sorted according to the alphabetical ordering (we define the word 'superior'; 'A' is superior to 'C', 'G', and 'T', etc.). In the sorted order, the beginning positions of the subsequences are written on an output file (e.g., 0.sort_p). Here, each subsequence is not actually copied onto the main memory; only the position pointers are generated and sorted according to the subsequences

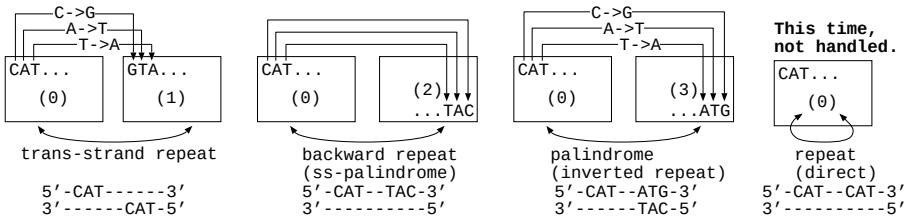


Figure 1: Converting the original sequence to detect palindromic or repetitive subsequences by comparing them with the original one.

pointed by these position pointers. Because the original sequence is fully stored on the disk, the subsequence can be restored if necessary. Here, the end of the sequence (EOS) is ordered last in the nucleotide order. This procedure (a) is applied to each converted sequence.

(b) To each combination for comparison (e.g., (0) and (3)), the position tables sorted by (a) (0.sort_p and 3.sort_p) are merged into one table (Fig. 2(right)): First, we open both table files. Secondly, the position according to the current line in each table provides the subsequence. This subsequence is not copied to other memory, but only virtually pointed to by the position in the full sequence stored in the memory array. Thirdly, we compare the current subsequence from 0.sort_p with the current subsequence from 3.sort_p and decide which is alphabetically earlier. The position of the superior subsequence is stored to the memory. After that, the current line of that side is incremented. Because each table is sorted, by repeating this procedure, we can make the tables merged into one table. This result is recorded in memory (see Fig. 2), and, the source pointer (source file ID) of each line is written in another memory array in the same order. In addition, the number of characters matched from the beginning with the subsequence of the previous line is calculated for each line and recorded in another memory array.

(c) Scanning each line of the row of the source of the result of (b), if we find a line from the converted sequence (i.e. 3), we search forward to the lines from the original sequence (i.e. 0): if we find a line of which the sequence is from the original sequence (0) and the number of coincidental characters with the sequence of previous line (column 'match') is not zero, it suggests that there is the subsequence in common with the original sequence and the converted sequence. Here, during this local search, the minimum value of the number of matched characters is updated (because

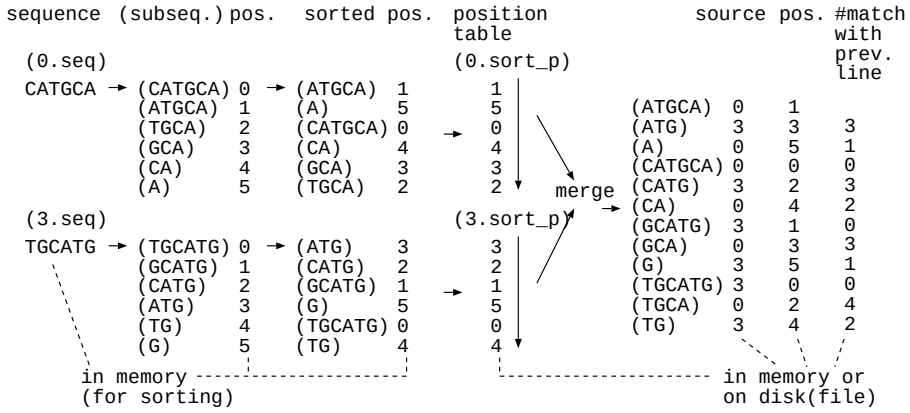


Figure 2: Sorting and merging the virtual subsequences (not in memory) of the original sequences. Only the full sequences and the position tables are in memory.

it depends on the comparison, it can not be determined previously). This minimum value is the true number of matched characters from the beginning between the original subsequence and the converted subsequence. If we find that the number of matched strings equals zero, then the search is discontinued because there will be no sequence matched with the sequence. For instance, in Fig. 3 (left), we start the global search from the first line until the last line to find subsequences which are from the converted sequence ('source' is 3). Because we can find the line signed (*) first, we start the local search from the next line (i) for finding subsequences which are from the original sequence ('source' is 0). We can find that the line (i) satisfies the condition. Because the number of matched characters between the subsequence 'A' indicated by the line (i) and the subsequence 'ATG' indicated by the previous line (*) exceeds zero (i.e., 1 indicated by the 'match' column of the line), we output this combination as a result candidate. We continue the local search. Seeing the next line (ii), we can find that the number of characters matched between the subsequence 'CATGCA' indicated by the line (ii) and the subsequence 'A' indicated by the previous line (i) does not match from the beginning (indicated by the 'match column of the line (ii) as 0). Thus we stop the local search. Let's return to the global search.

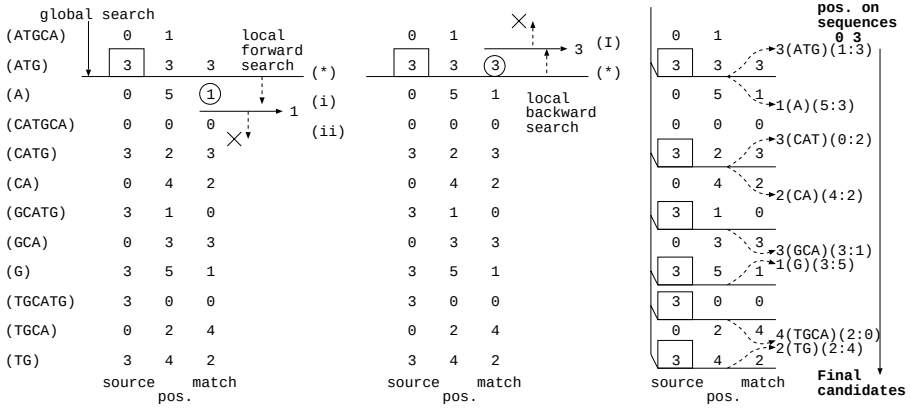


Figure 3: Searching matched subsequences: local forward search, local backward search, and final outputs by global search.

Sometimes we want to set the lower bound of the number of matched characters from the beginning of the subsequences. Then the local searches can be stopped earlier.

Simultaneously with the local forward searches, we also execute local backward searches from the key lines (e.g., (*) in Fig. 3 (middle)). Here, to see the number of matched characters from the beginning, we use the 'match' value of the next line. For example, at the local forward search from the line (*), the number of matched characters between 'ATG' indicated by (*) and 'ATGCA' indicated by (I) is 3 (shown at the column 'match' of the line (*), which is the next line of (I)). Thus, this combination of these lines is a candidate. Because (I) is the first line, we can not go backward any more. Although sometimes the local searches go beyond the next key lines, the search continues while the number of matched strings is above the lower bound.

All candidates picked here are passed to the post-processing part for refining results (Fig. 3(right)).

(d) We combine the sorted sequences processed at (a), i.e. (0) with (1), (0) with (2), and (0) with (3), and apply them with the procedure (b)-(c) respectively.

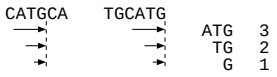


Figure 4: Redundant candidates (sub-subsequence pairs).

### *2.4 Post-processing the candidates*

First, because each position in the reversed sequences differs from the original one, the position is calculated with respect to the original position (the sequence length minus its value). Secondly, self-overlapping subsequences are omitted. Thirdly, because of the symmetricity of the palindromes and repeats, each candidate may have a redundant pair. To avoid this, only the subsequences of which position in the original sequence is after that in the converted sequence. Lastly, because we treat each part of subsequence to be different from the subsequence, the candidates are a mixture of combinations of sub-subsequences although their locations are the same as the subsequences. For example, let's consider sequence 'CATGCA' and 'TGCATG' (see Fig. 4). The subsequence 'ATG' appears in both the sequences. Also we can see at the same location 'TG' and 'G', which is sub-subsequence of 'ATG'. Because only after the pair is decided we can extract the matched characters, it is not possible to group them beforehand. Therefore, when the maximum length of the subsequence common in both the sequences is $M$, $M$ pairs will be produced as candidates. However, these redundant candidates have the same characteristic that the end of the matched characters have the same positions respectively. Thus, such redundant combinations can be gathered by checking the end position of subsequence in the original sequence ($A_{end}$), and that on the converted sequence ($B_{end}$); we can collect such combinations by sorting all of the combinations according to the position $A_{end}$, and by sorting candidates inside the group of the same $A_{end}$ position according to the position $B_{end}$.

## 3 Result

The program is written in C++ and is executable on a supercomputer (Sun Enterprise 10000, 64 processors). Although it can run on the computer in parallel, we evaluated it using a single processor. Because we implemented the merge process using disk file, the execution cost was larger than the on-memory implementation. However, it consumes only $5N$bytes (for the full sequence and the position table. See ahead). First, we applied our program to an eukaryotic promoter sequence (902bases) from EPD[3] setting the lower bound of the subsequence length to 7:

```
% palin EPD49023 7
— EPD49023, 7
 - trans-strand repeats -
  0:      8, (   265), [     6,    13], [   279,   286]:   GGGTAGGG
  1:      7, (    44), [   148,   154], [   199,   205]:   GGTTTCC
  2:      7, (   159), [   182,   188], [   348,   354]:   GTCTCCC
  3:      7, (   171), [   112,   118], [   290,   296]:   ACCCTCC
  4:      7, (    39), [   307,   313], [   353,   359]:   CCTTGGG
  5:      7, (   357), [    12,    18], [   376,   382]:   GGGTGCT
 - backward repeats -
  0:     10, (   150), [   287,   296], [   136,   127]:   AGGTGGGAGG
  1:      8, (    72), [   204,   211], [   131,   124]:   GGAGGAGG
  2:      7, (   204), [   423,   429], [   218,   212]:   GACGATC
  3:      7, (   160), [   296,   302], [   135,   129]:   GGTGGGA
  4:      7, (   159), [   404,   410], [   244,   238]:   CAATGGG
  5:      7, (   395), [   406,   412], [    10,     4]:   ATGGGGG
  6:      7, (   246), [   339,   345], [    92,    86]:   TTGAGGG
 - palindromes -
  0:      9, (    59), [   413,   421], [   353,   345]:   CCTCTGAGC
  1:      8, (     6), [   126,   133], [   119,   112]:   AGGAGGGT
  2:      7, (   174), [   261,   267], [    86,    80]:   CCTGCCT
  3:      7, (   100), [   299,   305], [   198,   192]:   GGGACTA
  4:      7, (    87), [   303,   309], [   215,   209]:   CTAGCCT
  5:      7, (    81), [   202,   208], [   120,   114]:   AAGGAGG
  6:      7, (   106), [   319,   325], [   212,   206]:   GCCTCCT
  7:      7, (    27), [   261,   267], [   233,   227]:   CCTGCCT
  8:      7, (   173), [   292,   298], [   118,   112]:   GGAGGGT
```

Here, ID number, subsequence length, spacer length, position in the original sequence, position in the converted sequence, and subsequence are indicated by each line. The execution time was 0.79 sec (total time includes memory access, disk access, printing process, and temporary-file removal). Indeed, we could extract long palindromic and repetitive subsequences. We also applied our method to the HPV16 DNA sequence (double strand, 16 Kbases). The execution time was 11.61 sec. Although we set a lower bound of the subsequence length to 7, we found this cancer-viral DNA includes 2689 trans-strand repetitive pairs, 2796 backward repetitive pairs, and 3150 palindromic pairs. The length of the longest subsequence is 12, 16, and 14 respectively.

## 4 Discussion

### *4.1 Related works and complexity*

Genomic science stands on the assumption that all information of the life is written in the full DNA and aims to decipher the cryptograms in it. In recent years, a large amount of genomic sequences are clarified and the function of each gene has been analyzed. Now we are to analyze the higher-order function, the global interaction such as intra-DNA binding affinity, the unit of each function in the DNA, and the protein-DNA bind-ability must be considered. In this research, first, the Nagao and Mori's algorithm[5] by which frequency statistics

of the character strings in the text itself has been expanded to an algorithm by which the substrings of the maximum length at each location common in two texts can be identified. Next, we resolved the problem of detecting repetitive subsequences such as palindromes into a problem of the comparison between two texts, and applied the above algorithm to this problem.

In this research, we generalized the concept of palindromic and repetitive sequences, and classified the phenomena according to the case when the same sequence (including opposite direction and on the complimentary chain) appears and how they function. Although only local areas in nucleic acids had been extracted to analyze their function, using this method, we can detect in one time the same subsequences at long distances. For example, to examine the transcription regulation, the range for analysis was only limited to the transcription regulatory area. It is the same situation in the analysis of the mechanism of the translation and the replication. However, recently, some people provide mechanisms for replication, repair, transcription, translation, signal transduction pathway, etc. that closely affect each other [7].

Early programs which detect palindromic sequences are only optionally developed for DNA sequence analysis. Thus, there is no tool that can handle the general situation of palindromic sequences using time and memory efficient algorithm. For example, although Bailey's program[2] also includes the option for detecting palindromes, because it only applies a complimentary symmetry matrix for each position, it can not detect the general palindromic sequences with spacers of arbitrary length. There is the problem that, once arbitrary spacers are considered, the amount of the calculation and memory required explodes. This research solved this problem.

Let's thinks about the ability of the implemented program. First of all, because we used 32 bit unsigned integer to represent the position in the input sequence, $2^{32} = 4Gbp$ can be treated (it also has information on a reverse-strand). This is enough to handle full sequence of the human genome. The amount of the calculation time of this algorithm is $O(N \log N)$ (Table 2). Here, $N$ is the number of nucleotides of the full sequence input. For sorting, we implemented comb sort. In the table, $\alpha$ is the number of combinations produced. This value affects the efficiency when the candidates are listed and compacted to the final results. If $N >= \alpha$, the time required for sorting subsequences ($O(N \log N)$) is critical for entire processing time. Otherwise, the time required for post-processing the candidates ($O(\alpha log \alpha)$) is critical. While the amount of the memory required is $O(N)$ (see the column at the center of the table), the memory used in each process is assumed to be reused by garbage collection. That is, the process in which the memory use is the largest in 1-6 becomes the total amount of memory required in this program. The four

Table 2: Order estimation of execution time and memory space (unsigned int = 4 bytes).

| Processing | Execution time | Memory space | Memory space (with disk) |
|---|---|---|---|
| 1. Sequence conversion | $O(N)$ | $N$ (unsigned int) | 1(unsigned int) |
| 2. Sorting subsequences | $O(N \log N)$ | $N$ (unsigned int) | $N$(unsigned int)+$N$bytes |
| 3. Merge | $O(N)$ | $2N$(unsigned int) | 1(unsigned int) |
| | | +$2N$bytes+$2N$bytes | +$(2 + L + O(1))$bytes |
| 4. Counting matched characters | $O(N)$ | $2N$(unsigned int) | 1(unsigned int) |
| | | +$2N$bytes+$2N$bytes | +$(2 + L + O(1))$bytes |
| 5. Generating candidates | $O(N + \alpha)$ | $2N$(unsigned int) | 2(unsigned int) |
| | | +$2N$bytes+$2N$bytes | +$(2 + L + O(1))$bytes |
| 6. Post-processing | $O(\alpha \log \alpha)$ | $\alpha$(unsigned int) | $\alpha$(unsigned int) |
| | | +$N$bytes | +$N$bytes |

unsigned integers used for each candidate in the post-processing are the temporary space for recording the beginning and end positions of the subsequences on both the original sequence and the converted sequence. If $N > (16/11)\alpha$, $2N \cdot$(unsigned int) $+4N$ byte used in the steps from 3 to 5 is the largest. On the other hand, if $N < (16/11)\alpha$, $\alpha \cdot 4$(unsigned int) $+N$ bytes used in the step 6 is largest.

Here, we can also consider the version which uses disk files in the steps 3-5, then, these processes hardly require memory. Although we recommend steps 2 and 6 are processed in memory because they are sorting, when the memory size is small, we can use the disk files using the merge sort algorithm. Even when using the disk, the execution speed and memory space are still in a trade-off relation. When the merge sort algorithm is applied, although we can expect high speed access with direct memory access transfer (block transfer from the disk to the main memory), temporary memory for processing is also required. For example, merge sort with ideal memory space requires only $O(\log N)$ file access. However, it requires $N$ memory space and $O(N \log N)$ processing time (including the memory access frequency). The total execution time is the summation of the file access time, the memory access time, and the calculation time. Under smaller memory size, as the frequency of the block transfer increases, a file access frequency more than $O(\log N)$ is needed, which reduces the efficiency.

### *4.2 Limitation of our algorithm*

Our algorithm aims to extract completely matched subsequences. Thus, neither the mismatch nor gaps in the subsequences are considered although spacers between the subsequences were considered. In general, it is difficult to estimate how mismatches or gaps affect the binding affinity or conformation structure of nucleic acids because they largely depend on the entire structure.

Therefore, we focused our task to the simplest case: extracting completely matched subsequences as the 1st approximation. In future work, we have planned to assemble the subsequences extracted by our algorithm to handle the mismatch and gap problem. Our algorithm for extracting palindromic and repetitive subsequences will be the first step to the processing of the grammatical structure of nucleic acid sequences.

## 5 Conclusion

Up to now, the meaning of palindromes and repeats has not been looked into deeply. Especially, backward-repeats and trans-strand-repeats from which the possibility of binding was thought to be negligible. However, recent investigations suggest that a lot of unexpectedly long sequences are commonly seen separated by large distances in nucleic acids. Thus, we proposed the cost and memory efficient algorithm to extract such sequences. With the current computer ability, a large amount of genomic sequences, and the data of gene expression in many different situations, the relation between the function and the structure of nucleic acids converges through structural biology.

## Acknowledgements

This work is partially supported by Grant-in-Aid for Scientific Research on Priority Areas, "Genome Science" from the Ministry of Education, Science, Sports, and Culture, Japan.

## References

1. B.Alberts et al. *Molecular biology of the cell. Third ed.* (1994), 463-468.
2. T.L.Bailey. Discovering motifs in DNA and protein sequences. *Univ. of. California at San Diego (Ph.D. dissertation)* (1995).
3. R.Cavin, T.Junier, and P.Bucher. The Eukaryotic Promoter Database EPD, R.50 (1997). Swiss Institute for Experimental Cancer Research.
4. RA.Marciniak et al. HIV-1 Tat protein trans-activates transcription in vitro. *Cell* (1990) **63(4)**, 791-802.
5. M.Nagao and S.Mori. A New Method of N-gram Statistics for Large Number of n and Automatic Extraction of Words and Phrases from Large Text Data of Japanese. *In Proceedings of International Conference on Computational Linguistics* (1994), 611–615.
6. A.Tropsha et al. Making sense from antisense: a review of experimental data and developing ideas on sense–antisense peptide recognition. *J.Mol.Recognit.*(1992) **5(2)**, 43-54.
7. TBP held hostage by cisplatin. *nature structural biology* (1998), Vol.5, 103.