DNA사슬 내에서 다양한 길이의 팰린드롬쌍 검색 연구

김 형 래[†] · 정 경 희[††] · 전 도 홍[†††]

요 약

게놈 프로젝트 연구는 DNA사슬 내에서 생물학적 의미(예, molecule 의 진화역사 또는 그 기능)를 추출하기위한 사슬분석 쪽으로 강조가 되어가고 있다. 특히, DNA사슬 내에서 상보적 또는 반복되는 패턴은 생물학적 의미를 가지고 있다. 문제는 상보적 단어가 만들어내는 흥미 있는 패턴과 단어 구성을 찾아 내는 것이다. 본 논문은 다양한 길이의 팰린드롬 쌍을 검색하는 알고리즘에 관한 연구이다. 다양한 길이의 팰린드롬 쌍 내에는 빈 공백을 또한 허용한다. 알고리즘은 팰린드롬 알고리즘이라고 명명하며 O(N)의 계산 시간을 가진다. 하나의 팰린드롬 쌍은 머리핀 형태로 구성되어 있다. 검출된 여러 팰린드롬 쌍을 활용하여 n-쌍 팰린드롬 형태를 구성하였다. 더욱이 발견된 가장 긴 팰린드롬 쌍을 DNA 사슬 원형 구조에 점으로 표현하여 가시성을 제고하였다. 본 알고리즘은 여러 게놈 상에서 실시되었으며 E.coli K12 의 결과를 나타내었다. 실험결과 DNA 안에는 랜덤한 경우에는 확률상 매우 발생하기 힘든 긴 팰린드롬 패턴들이 존재 한다는 것을 발견할 수 있었다.

키워드 : 팰린드롬 쌍, 상보적 패턴, 유전자 알고리즘, DNA 구조, 사슬분석 알고리즘

Identifying Variable-Length Palindromic Pairs in DNA Sequences

Hyoung-Rae Kim[†] · Kyoung-Hee Jeong[††] · Do-Hong Jeon[†††]

ABSTRACT

The emphasis in genome projects has Been moving towards the sequence analysis in order to extract biological "meaning" (e.g., evolutionary history of particular molecules or their functions) from the sequence. Especially, palindromic or direct repeats that appear in a sequence have a biophysical meaning and the problem is to recognize interesting patterns and configurations of words (strings of characters) over complementary alphabets. In this paper, we propose an algorithm to identify variable length palindromic pairs (longer than a threshold), where we can allow gaps (distance between words). The algorithm is called palindrome algorithm (PA) and has O(N) time complexity. A palindromic pair consists of a hairpin structure. By composing collected palindromic pairs we build n-pair palindromic patterns. In addition, we dot some of the longest pairs in a circle to represent the structure of a DNA sequence. We run the algorithm over several selected genomes and the results of E.coli K12 are presented. There existed very long palindromic pair patterns in the genomes, which hardly occur in a random sequence.

Key Words : Palindromic Pattern, Algorithm, DNA Sequence, DNA Structure

1. Introduction

Motivation: One of the problems arising in the analysis of biological sequences is the discovery of patterns that appear at different positions in a nucleic acid. Genomic science and structural biology have relationships in terms of the sequence and the structure of nucleic acids. It is well known that in a palindrome of nucleic acids the subsequence binds with the subsequence in the opposite direction complementary on its own strand to make a stem-loop. Since in DNA topological entanglements such as knots and catenation are crucial to the function of cells, it is important to find these palindromes and knots [1]. Especially, palindromic or direct repeats that appear in a sequence have biophysical meaning [18]: recognition site of dimers, forming stem-loops, and contributions to global structure of nucleic acids; moreover, the genetic network, transduction pathway, and tissue specificity are also related to these sequences.

Problem: Our research focuses on finding inverted repeats (palindromes). When we start to search for palindromic pairs in a DNA sequence, we basically have

† 정 회 원 : 한국고용정보원 정보화기획팀 부연구위원
†† 준 회 원 : 관동대학교 전자계산공학과 박사과정
††† 정 회 원 : 관동대학교 컴퓨터학과 교수
논문접수 : 2007년 6월 20일, 심사완료 : 2007년 8월 29일

little information about the words (subsequences) for which we are searching. We can find variable length palindromes longer than the minimum word length (this can be calculated either automatically or assigned by a user). Palindromes may contain some mismatches in the form of gaps and defects of various other natures [4,7]. So we let the algorithm allow errors within a word. We may be able to compose these collected palindromic pairs (that consists of a hairpin structure) to multiple pair palindromic patterns. Furthermore, we can use this algorithm to understand the structure of a DNA sequence.

In order to study further how to use the variable length palindromes, we can adapt them to identify pair pin structure. But, there exist many long palindromes in a DNA sequence, which hardly occur in a random string. We assume that these long palindromes can represent more complicated structures.

Approach: In order to find palindromes, we propose a new algorithm that break a string into smaller subset of the string and then merge them to be a real one. We name this searching technique as "Break and Gather". The palindrome-searching algorithm (PA) divides a sequence into n-gram tokens, and then combines tokens by merging adjacent tokens to long words. While merging two adjacent tokens, we allow errors by using BLAST [14] technique.

If we have a multiple hairpin structure, we can easily extend it into two and three pair palindromic patterns. Moreover, once we pick a certain number of palindromes (where n can be any number such as 30, or 50), we can also identify a certain type of them. Another simple way to represent these palindromes is to plot them on a circle. We hope our new structural representation scheme, which uses n-pair palindromic patterns, would help biologist in visualizing and understanding DNA sequences.

As an advantage, this algorithm has almost linear time complexity O(N×LTK) for the length of a DNA sequence and O(N×LTK) space complexity, where LTK is n-gram window size. The disadvantage of PA is it is not complete and may not find a palindromic pair even if one exists. This, however, happens only in a very synthetic string sets (for example a string that consists of only one character).

The contributions for this work are:

(1) we introduce a new palindrome searching algorithm (PA) that uses "Break and Gather" technique;
(2) we extend a palindromic representation from hairpin structure to multi-pair palindromic patterns, introducing a new possibility of structural representation of a DNA sequence;

In section 2 we discuss related works in searching palindromes and representing DNA sequences; Section 3 introduces palindrome and palindromic pattern, and then describes inputs and outputs, Section 4 details our palindrome algorithm (PA); Section 5 describes the experiments; Section 6 analyzes the results from the experiments; Section 7 summarizes our work and suggests possible future work.

2. Related Works

Bailey [17] developed a program that can detect palindromes; but, because it only applies a complementary symmetry matrix for each position, it could not detect the general palindromic sequences with gaps of arbitrary length. In his algorithm, the time complexity and memory complexity explode, when arbitrary gaps and lengths were considered. There is an algorithm that can find in linear time the longest substring of string s whose reverse is also a substring of s [4]. That algorithm cannot be generalized to report all palindromic pairs longer than a given threshold like our algorithm. It applies a fixed gap between a word and its complement. If we generalized that algorithm to allow various gaps, the complexity will be explored. To solve the proposed problem Tsunoda [18] devised an algorithm that extracts variable length palindromes allowing variable length of gaps. They classified repetitive patterns as four cases: (1) direct repeats, (2) trans-strand repeats, (3) backward repeats, (4) inverted repeats (palindrome). They clearly described the necessity of various types of the repetitive patterns. Their algorithm searched for patterns of those four types with O(Nlog$N$) time complexity, where N was the size of sequences; they found exact matches. Porto [2] allowed errors within palindromes; that is, they introduced approximate palindromes. Their algorithm found all approximate complements with K errors of a certain word. The difference is that our algorithm finds a complement for each word. BLAST [14] directly approximates alignments that optimize a measure of local similarity; this uses maximal segment pair (MSP) score. The MSP is adapted to PA for allowing some errors while merging adjacent words. It takes O(n^2) for BLAST to search all palindromes in a string [15]. There have been parallel palindrome-searching algorithms that find all

palindromes of a certain initial word allowing complement gaps [0,3]. PM, however, takes O(n) time complexity.

Jensen [10] presented a method for visualizing complete microbial chromosomes so that repetitive sequences (direct repeats, inverted repeats etc) or DNA structure became visible. It is difficult to get an overview of a complete genome due to its size. They used an atlas (a colored circle) for representing sequences and structures. In our research we plotted palindromic pairs in a circle that represents a whole sequence. We hope this can give an insight towards the symmetric of a genome. Eisen [6] found symmetry around the replication origin and terminus. They also found statistically significant x-shaped patterns within some genomes, indicating that there is symmetry about the replication origin. However in our structural representation and plotting of palindromic pairs, clear symmetric shape is not found.

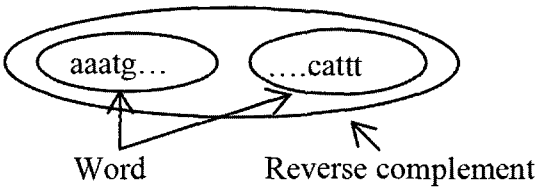
3. Palindromic Pair and Patterns

Palindromic pairs represent hairpin structures (stem-loop formation) that have biological meaning in DNA and RNA [18]. A palindrome is an inverted repeats of a word (subsequence) - a word is a sequence of characters. When searching a DNA sequence for this palindromic pair, we do not have the information about the word length or how many palindromic pairs exist in a DNA sequence. Even we do not know which length of a palindrome has biological meaning. Therefore, Tsunoda et al. [18] searched for variable length palindromic pair, as we do. A palindromic pair can represent only a hairpin structure. With multiple palindromic pairs we can extent them to multi-pair palindromic patterns.

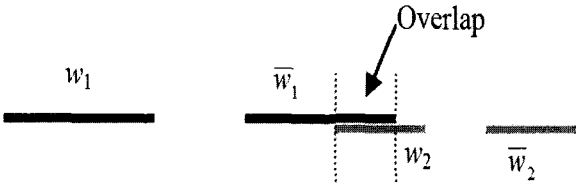
3.1 Palindromic pair

A palindromic pair consists of a word and its reverse complement as shown in (Fig. 1). Even though its structure is not complex, there are some characteristics: gaps, length, overlap, and errors. There can be a gap between the word and its complement. Palindrome recognition differs according to the presence of a gap and differences in length; it is necessary to examine complements with gaps [18].

There can be overlaps among palindromes as shown in (Fig. 2). Palindrome (w_1 and $\overline{w}_1$) and (w_2 and $\overline{w}_2$) are overlapped. There can be many ways of collecting pairs from this situation: first, collecting only the first pair w_1 and $\overline{w}_1$; second, collecting the longer pairs; third,



(Fig. 1) Palindrome



(Fig. 2) Overlapping palindromes

collecting both pairs; and fourth collecting the second pairs w_2 and $\overline{w}_2$. Since one of our purposes is to collect as many pairs as possible, we chose the third method that collects both pairs. In some cases three or more pairs can overlap each other. However, since the biological meaning of overlapping is not known yet, we do not collect all possible pairs allowing multiple overlaps (we only allow overlap one time). Once we get the palindromic pairs including overlapped pairs, those overlapped pairs can be removed easily depending on the necessity.

There can be mismatches in the form of gaps and defects in a sequence itself [4,7]. Palindrome searching algorithm (PA) should allow error or mismatches between words and complements.

3.2 Palindromic pattern

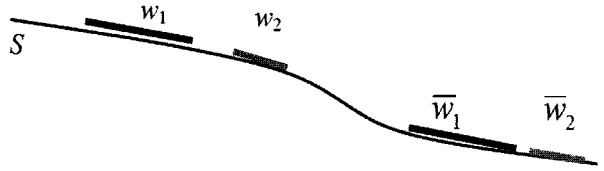
A palindromic pair constructs hairpin structures; once we get multiple palindromic pairs we can easily extend the structure to multi-pair palindromic patterns by composing multiple palindromic pairs.

Two words w_1 and w_2 with complements $\overline{w}_1$ and $\overline{w}_2$ form 2-pair palindromic patterns. There are three different 2-pair patterns, which can be identified by integers or ordinal quadruples: (1,1,2,2), (1,2,1,2), and (1,2,2,1). (Fig. 3) presents three types, where each number inside the bracket represents a word; the repeated numbers (on the right side) represent their complements.

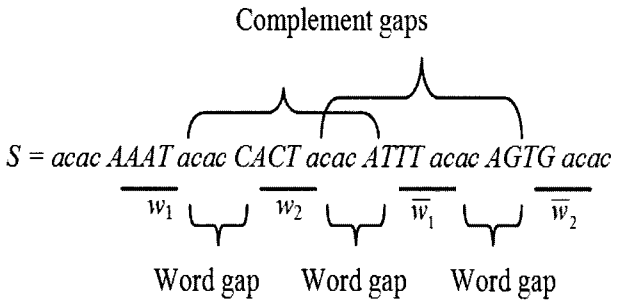
These types can also be represented in a sequence S. For example the type of (1,2,1,2) that is identical to (w_1, w_2, $\overline{w}_1$, $\overline{w}_2$) can be represented in a string S as shown in (Fig. 4).

Type 0: (1, 1, 2, 2)
Type 1: (1, 2, 1, 2)
Type 2: (1, 2, 2, 1)

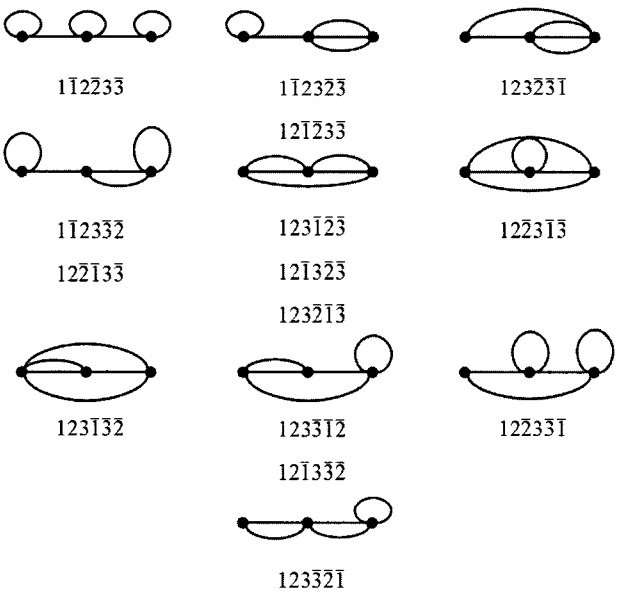
(Fig. 3) 2-pair palindromic patterns



(Fig. 4) Pattern type 1 in a string S



(Fig. 5) Word and complement gap



(Fig. 6) Similar pattern types in 3-dimensions

Two pair palindromic patterns also have restrictions just as 1-pair palindromic patterns do, yet another constraints can be applied - a minimum and maximum gap between words. Words should not be closer than the minimum word gap and should not be separated by more than the maximum word gap. We call the gap between a word and its complement "complement gap" and the gaps between words "word gap". These are depicted in (Fig. 5), where capital letters represent words and their complements and lower letters represent any words.

N-pair palindromic patterns can be generated depending on the n. For example there can be generated three different types of 2-pair palindromic patterns and 15 different types of 3-pair palindromic patterns. More dimensional palindromic patterns will generate more numbers of types. Some of their patterns are isomorphic when viewed as a graph. By disregarding directions some patterns form the same shape. For example the two patterns (1,1,2,3,2,3) and (1,2,1,2,3,3) become one type by graph similarity as shown in (Fig. 6). Once we pick certain number of palindromic pairs (e.g., 30) that is collected in a DNA sequence, we may be able to represent a type of n-pair palindromic pairs, which is called "structural representation" in this paper.

3.3 Input/output data

Our research first focuses on collecting palindromic pairs in a DNA sequence, and then extend to 3-pair palindromic pairs and structural representation of DNA sequence with the collected pairs.

In collecting palindromes the palindrome-searching algorithm (PA) is used, several options (constraints) can be assigned. First, we may need to set the token (n-gram) length. Sometimes we may only want long palindromic pairs, so we need to set the minimum length. We can assign minimum and maximum complement gaps (the gap between a word and its complement) and minimum and maximum word gap (the gap between words). The values for these gaps can be any value bigger than or equal to 0 and less than the sequence length. The error rate means the number of the allowed mismatches when tokens are combined. The output is a set of variable length palindromes. An example is shown in (Fig. 7). We list all options for PA and assign their values arbitrarily. The lengths of output palindromes are different; we try to collect palindromes with the possible maximum length.

We extend those collected palindromes to multi-pair palindromic patterns. A palindrome constructs hairpin structure; by composing multiple palindromes we can build more complicated structures - multi-pair complimentary patterns.

Input:
A DNA sequence: "…aaattaata … .. aataaaaaga … gataaact … .. tattaattt … .. tctttttatt … …. .. agtttatc …"
Token length: 8
Minimum length of a palindromic pair: 9
Minimum complement gap: 10
Maximum complement gap: the size of a sequence
Minimum word gap: 10
Maximum word gap: the size of a sequence
Allowed error: 0

Output:

Word		Complement	
Position	String	Position	String
23	aaattaata	358	tattaattt
50	aataaaaaga	761	tctttttatt

(Fig. 7) Example of input/output for PA

4. Palindrome and Pattern Searching

Palindrome searching algorithm(PA) detects palindrome with variable length and gaps from a DNA sequence. When the objects that we are searching for are not known and lie in an enormous of data, the problem will be difficult. For this kind of a problem we propose a new searching method, called "Break and Gather". It breaks the problem of finding long palindromes into finding small, fixed-sized chunks, then gathers and combines the chunks that can be combined. With those collected palindromes by PA, we extend to multi-pair palindromic pairs, and count all kind of existing types of n-pair palindromic patterns.

4.1 Palindrome algorithm

We use n-gram tokens to search a DNA sequence for words. At first we set the length of a token (LTK). We treat characters as numbers. Each token is encoded and stored in an open hash table, an array of pointers shown at line 1 in (Fig. 9). As we store a token, the existence of its complement is also checked shown at line 3. We do not convert each token to its complement every time a token is inserted into the hash table; instead, we pass the complement together with the token. The complement gaps and word gaps are considered while searching for a complement token. All the token pairs that meet the constraints are retrieved from the DNA sequences. If a valid token pair that meets complement gaps is found, it will be stored in a token table (TT) that holds the token, the complement token, and their positions shown at line 4. The structure of a token table is shown in (Fig. 8).

Once the whole DNA sequences are scanned, the token table is sorted by positions. When all the pairs are sorted by position some tokens and complements are adjacent to the next token and complement. It is clear that those pairs can be combined. Therefore, we combine adjacent tokens and store them in a combined token table (CTT) shown at line 9. Out of the combined token table we select and return only the combined tokens longer than minimum word length and store them in a word table shown at line 10. Pseudo code is shown in (Fig. 9). The following sections explain the details.

Positions	Pointer of a token	Positions	Pointer of the complement

(Fig. 8) Structure of the token table

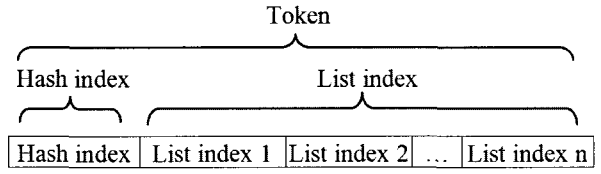
```
Sequence: DNA sequence
tLength: the length of a token
          /* 11 in our experiment */

Procedure PA (Sequence, tLength)

1. OpenHash<-BuildOpenHashTable(tLength)

  Step1: scan the Sequence and build TokenTable
2. While (Token←ReadToken(Sequence))
3. if Token exists in OpenHash then
4. TokenTable←Store Token and its Reverse
      Complement Token in the OpenHash
5. else
6. OpenHash←Token
7. end if
8. end while

Step 2: sort TokenTable by position and combine
     adjacent tokens in TokenTable
9. CombinedTokenTable←CombineAdjecentTokens
                          (TokenTable)
10. Return CombinedToken which is longer than
          minimum length from CombinedTokenTable

End Procedure
```

(Fig. 9) Palindrome algorithm

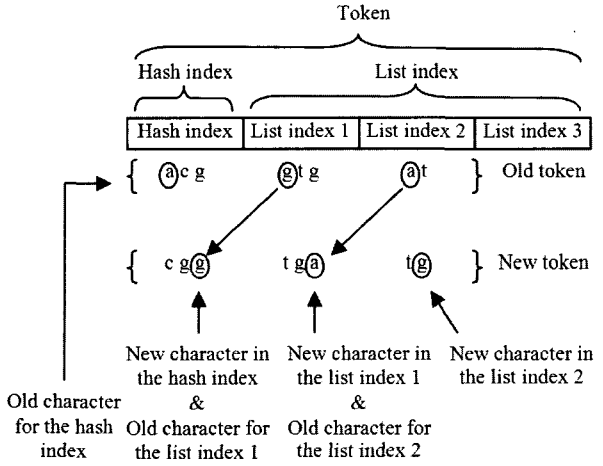
4.2 Token structure

We break a DNA sequence into small tokens and start to search for complementary token pairs. We treat the characters as numbers (a=0, c=1, g=2, and t=3). The problem is how to convert a long set of characters into a number. For instance, a token v="caacgt" will be calculated like this: $1\times4^5+0\times4^4+0\times4^3+1\times4^2+2\times4+3=1051$. Since we are converting characters into a number there

is a limitation in the token size. Suppose we are using a signed long integer in C++, where the maximum value is 2,147,483,647. The maximum number of characters that can be encoded will be easily calculated as 15 (4^{15}<2,147,483,647<4^{16}). However the length of a token (LTK) could be longer than 15. So we divide a token into two parts: a hash index and list indexes as shown in (Fig. 10). The first 11 characters are used for the hash index and the remainders are grouped into 15 character chucks, except the last chunk that may be smaller. For example, when a token length is 50, the first 11 characters are used for the hash index and the next 15 characters are used for the first list index, the next 15 ones for the second list index, and the last 9 characters for the third list index.



(Fig. 10) Structure of a token



(Fig. 11) Converting character to number

If we encode every tokens independently the time complexity will be O(LD×LTK), where LD is the length of the sequence and LTK is the token length. We view a token v of c consecutive characters as representing a length-c quaternary number. Given a string $T[1..d]$, let t_s denote the quaternary value of the length-c substring $T[s+1..s+c]$, for $s=0,1,\cdots,d-c$. Certainly, $t_s=v$ if and only if $T[s+1..s+c]=v[1..c]$. We can compute the value of token v in time $O(c)$:

$$V=P[c] + 4(P[c-1] + 4(P[c-2]+\cdots+4(P[2]+4P[1])\cdots)) \quad (1)$$

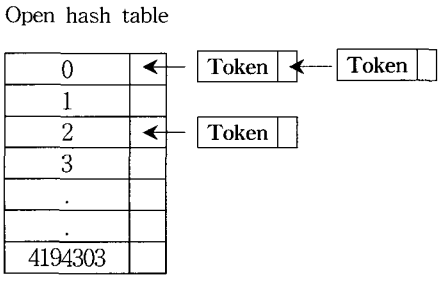
where V is the computed numeric value of token v. To compute the remaining values $t_1,t_2,\cdots,t_{d-c}$ in time $O(d-c)$, it suffices to observe that t_{s+1} can be computed from t_s in constant time [16], since

$$t_{s+1} = 4(t_s - 4^{c-1} T[s+1])+T[s+c+1]. \quad (2)$$

For example, let us say a token is "acggtgat", and the next token is "cggtgatg". The token length is 8; the hash table length is 3; and the length of each list word is 3. First, "acg" is encoded and 6 ($=0\times4^2+1\times4^1+2$) is kept as the hash index, "gtg" is encoded as 46 and kept as list index 1, and "at" is encoded as 3 for list index 2. The first character "a" of the hash index in the token becomes the old character for the new hash index while the "g" in list index 1 becomes the new character in the hash index. Similar shifts occur for the other indices, which are depicted in (Fig. 11). The new hash value is 26 ($=4\times(6-0\times4^2))+2$), where 6 is old value, 0 is old character ("a"), c is index length (3), and 2 is new character ("g").

4.3 Hash table

While scanning a DNA sequence each token is stored in an open hash table. Since we are trying to build a token table (a table that stores tokens and their complements with positions), every time a token is scanned the existence of its complement should be checked also. And if it is found, both should be stored in the token table. The structure of a open hash table is basically just an array of pointers to linked lists that stores a token. The structure is shown in (Fig. 12). In our experiment, the number of element stored in the hash table is 4,194,304(=4^11), where the length of a token is 11 and the number of different characters (a, c, g, t) is 4. The optimal size of the hash table depends on the size of a DNA sequence. If the size of hash table is too small the searching speed will decrease.



(Fig. 12) Open hash table

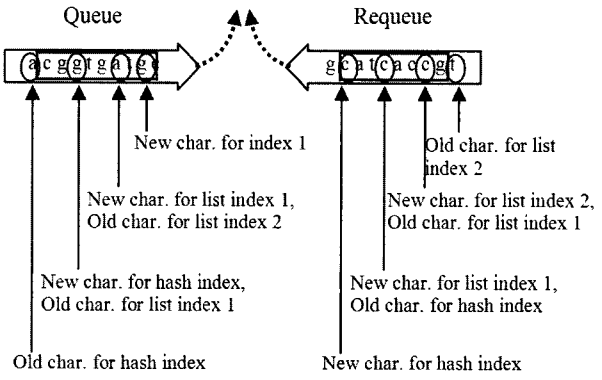
4.4 Finding a complement token

We explain how to find complement tokens when tokens are stored in a hash table. We do not convert each token to its complement and search the hash table; instead, we pass its complement together with the token. First let us explain how to know new added or removed characters from a token with constant time complexity. We devise a special type of queue, called an open queue, which removes the first element and get a new element when it is full, and has a Peek method that sees the character at a certain position besides the standard Enqueue and Dequeue methods.

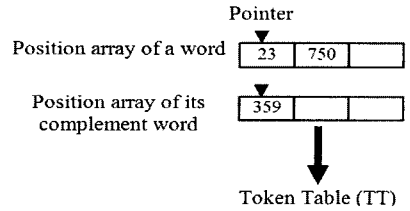
Two queues are made, whose size is bigger than the token length by one with open queue type: one is called queue and the other is called requeue. While scanning, all the words pass through the queue and all complement token through the requeue. One grows from the right and the other from the left as shown in (Fig. 13) The time complexity of passing through or peeking into a queue is O(1). By peeking certain positions new and old characters for each section of a token are picked.

While searching for a complement token, the constraints are considered: word gaps and complement gaps. All the token pairs that meet the constraints are retrieved from the sequence. If a valid token pair is found, then it is saved in a token table (TT) that holds the token, the complement token, and their existing positions. Each token instance has a position array that stores all positions where the token occurred. When a position is added to this array, a check to the complement token position array is made. if there is a valid pair, than the position pair is sent to the token table. After this, the array index of the smaller position value is advanced so that position will not be used for future comparisons in pair searching.

For example, in the case of (Fig. 14), the first position value (23) arrives and is stored in the position array of the token. In other words, a token "aaattaata" appeared at



(Fig. 13) Open queue and open requeue



Word		Complement	
Position	String	Position	String
23	aaattaata	359	tattaattt
50	aataaaaaga	761	tcttttatt

(Fig. 14) Finding a token pair

the position of 23 in the DNA sequence and the token is stored in the Hash table with its position information. At this moment no comparison occurs since the reverse complement token's position array is empty - its reverse complement is not appeared yet. When its reverse complement, "tattaattt", arrives later at the position of 359, it becomes a pair with the position (23). The pair (23, 359) is sent to a token table, and the pointer at the position of 23 increases. When the token "aaattaata" appears again at the position of 750, it pairs with 359, and the pair (359, 750) is sent to the token table.

4.5 Combining adjacent tokens

The method of breaking a huge string into small parts and locating complementary token pairs was explained in the previous section. Now we explain how to combine the parts. When all the elements in a token table are sorted by positions, some tokens and complements are adjacent to the next token and complement. It is clear that those pairs can be combined. In the example of (Fig. 15), the token and complement token at the positions of (23, 24) and (359, 358) respectively overlap except one character. We can combine these into one larger token - this procedure is called combination. All adjacent token pairs are combined and the results are stored in a combined token table (CTT).

The positions of next token can be represented as p+1, when p is the position of a token without allowing errors. But when we allow E errors (E is the number of error characters), the position of next word becomes between $p+1$ through $p+1+E$. Various techniques can be applied for this error counting such as maximal segment pair (MSP) score that is used in BLAST [14].

The combined tokens are not considered as words yet. Only the ones whose lengths are bigger than the minimum word length are stored into a word table (WT). For instance if the minimum word length is 9, the token at the position of 100 is not selected for the word table.

Token table (TT)

Word		Complement	
Position	String	Position	String
23	aaattaat	359	attaattt
24	aattaata	358	tattaatt
50	aataaaaa	763	tttttatt
51	ataaaaag	762	ctttttat
52	taaaaaga	761	tcttttta
100	gataaact	932	agtttatc
	...		...

Comibined token table (CTT)

Word		Complement	
Position	String	Position	String
23	aaattaata	359	tattaattt
50	aataaaaaga	761	tctttttatt
100	gataaact	932	agtttatc
	...		...

(Fig. 15) Combining adjacent tokens

5. Experiments

The purpose of our research is to identify palindromes globally. Experiments were conducted on the DNA sequences of Escherichia coli K12 (E.coli K12) [13], Escherichia coli o157 (E.coli o157) [12], and Salmonella [11]. We present the results from E.coli K12 (ACCESSION U00096LOCUS ECOLI 4639221 bp DNA circular BCT) in this paper. It is because the purpose of this paper is more related to the computation than the biological results. The results of the rest sequences can be provided upon request. The size of those genomes is around 5MB. The computer that we used is UNIX Sun Ultra 60 Workstation, and this program is implemented in C++.

We do not assign any constraints on finding palindromes over E.coli K12. The options are the token length (50), minimum length of a palindromic pair (50), minimum complement gap (0), maximum complement gap (without limitation), minimum word gap (0), and maximum word gap (without limitation). We did not assign any constraints to the gaps and errors to simplify our experiment. The base counts were 1,142, 136 for 'a', 1,179,433 for 'c', 1,176,775 for 'g', and 1,140,877 for 't'. There are no other characters other than those four base characters.

6. Analysis

Time complexity is one of the main evaluation criteria. We listed the time complexities of main processes in <Table 1> The time complexity for scanning and building a token table is O(LD), where LD is the size of a DNA sequence; the space complexity is O(LD×LTK), where LTK is the length of a token. The space complexity can be reduced by storing pointers of the tokens. Sorting the token table takes O(TT log TT), where TT means the size of a token table. Generally, TT is much smaller than LD, so the time complexity remains O(LD). CTT is also much smaller than the TT and LD, where CTT means the size of combined token table. We, therefore, can still claim that the time complexity of this algorithm is O(LD). Tsunoda [18] introduced an algorithm with O(LD log LD), which is slower than PA.

<Table 1> Main complexity

Processing	Execution time	Memory space
1. Scanning and building token table	O(LD)	O(LD×LTK)
2. Sorting the token table	O(TT log TT)	O(TT)
3. Building adjacent token table	O(CTT)	O(CTT)
4. Collecting palindromic pairs	O(CTT log CTT)	

* LD is the size of a DNA sequence
* LTK is a token size
* TT is the size of token table
* CTT is the size of a combined token table

The following describes the results of E.coli K12. The longest palindrome was 1,456, which is unusual in random sequences. The number of word pairs with length 986 and longer was still 14. Other results are listed in <Table 2> The results from E.coli K12 were surprising because there were many very long palindromic pairs, whose size is very unusual in a random sequence. In a random sequence with a length of 4,639,221, the maximum word pair length is 22 ($\log_2$4,639,221). We confirmed this by testing with three randomly generated sequences that have similar sizes.

We also counted the number of 3-pair palindromic patterns with respect to 15 different types. The frequencies of pattern type 0 through 14 are listed in <Table 2> The bigger number denotes the domination of a pattern type. Type 13 is relatively less dominant then the others, but it is very difficult to figure out the meaning.

We put similar patterns together and showed the results in <Table 3> We have not figured out the meaning yet. We hope this would help biologists understand the structure of a DNA sequence.

We chose 30 longest palindromic patterns as shown in <Table 4> The number 30 was arbitrarily picked. The word length, and start and end positions of a word and their complements are listed from the left. The end position of a word and its complement can be calculated

〈Table 2〉 Summarized results of E.coli K12

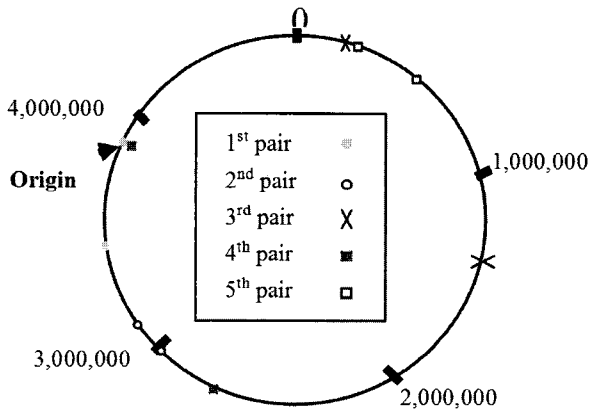
Description	Value
Token length	50
Minimum length of a palindromic pair	50
Minimum complement gap	0
Maximum complement gap	Without limitation
Minimum word gap	0
Maximum word gap	5,000,000
Total DNA sequence	4,639,221
a	1,142,136
c	1,179,433
g	1,176,775
t	1,140,877
others	0
longest pair length	1,456
longer than or equal to 986	14
Between 518 and 985	17
Between 50 and 517	206
Total palindromic pair	237
Pattern 0, 112233	153,488
Pattern 1, 121233	146,737
Pattern 2, 122133	232,621
Pattern 3, 122313	207,061
Pattern 4, 122331	102,535
Pattern 5, 112323	225,227
Pattern 6, 121323	111,547
Pattern 7, 123123	95,426
Pattern 8, 123213	95,955
Pattern 9, 123231	88,033
Pattern 10, 112332	220,651
Pattern 11, 121332	85,993
Pattern 12, 123132	106,084
Pattern 13, 123312	67,941
Pattern 14, 123321	140,977
Total Pattern	2,080,276

〈Table 3〉 Combine similar pattern types in E.coli K12

Description	Value
Pattern 0, 112233	153,488
Pattern 3, 122313	207,061
Pattern 4, 122331	102,535
Pattern 9, 123231	88,033
Pattern 12, 123132	106,084
Pattern 14, 123321	140,977
Pattern 1,5 (112323,121233)	371,964
Pattern 2,10 (122133,112332)	453,272
Pattern 6,7,8 (121323,123123,123213)	302,928
Pattern 11,13 (121332,123312)	153,934
Total 3-pair palindromic pattern	2,080,276

〈Table 4〉 Long word pairs in E.coli K12

Index	Word length	Word		Complement	
		Start position	End position	Start position	End position
1	1456	3422557	3424012	3941735	3943190
2	1332	2994383	2995714	3184111	3185442
3	1331	380484	381814	1465934	1467264
4	1314	2725483	2726796	3941735	3943048
5	1265	390932	392196	565994	567258
6	1260	565999	567258	1093467	1094726
7	1259	1093468	1094726	2168192	2169450
8	1255	314453	315707	390933	392187
9	1221	1467320	1468540	4505027	4506247
10	1204	687069	688272	1394064	1395267
11	1203	573810	575012	1394064	1395266
12	1195	273179	274373	1394068	1395262
13	1141	2066962	2068102	3184303	3185443
14	1058	227494	228551	2724412	2725469
15	834	2064488	2065321	3128215	3129048
...	...	...	...	...	...



(Fig. 16) Word pairs dotted in a circle

by adding the start position and word length. These 30 pairs can identify 6.19×10^{15} different types (Eq. 3). Instead of indexing by the order of palindrome length, we changed the index with respect to the position of words. We call this sequence a 30-pair palindromic pattern (structural representation of E.coli K12).

1 2 1 3 4 4 5 5 6 7 8 9 9 7 10 6 11 11 12 13 8 12 13 14 3 10 15 14 16 17 18 19 17 19 20 21 22 20 23 22 21 16 18 24 25 26 27 23 15 28 29 26 29 27 30 25 30 28 24 2

As we got this structural representation from other two genomes (E.coli o157 and Salmonella), we could observe that they are different from each other. Even

though we could not prove that this value can identify genomes theoretically, it appeared that different genomes have different structural representations.

Jensen visualized complete microbial chromosomes in a circle so that repetitive sequences in base composition or DNA structure become visible [10]. We borrowed their idea and presented some longest pairs to see the DNA structure. Since, dotting all the palindromic pairs can confuse us, the five longest palindromes were represented in a circle in (Fig. 16). One of the biggest surprises in genetics is a loop in mRNA. It seems that Eucaryotic genes contain loads of junk messages - non-coding sequences. The genes themselves are fragmented [5]. E.coli K12 is not a Eucaryotic genome but has similar loops. The palindromic pairs 1st, 2nd, and 5th are close together; therefore, matching them can generate small loops. Genes are symmetric about the origin of replication that indicates that matching sequences tend to occur at the same distance from the origin [6]. However, the circle with the five longest word pairs in (Fig. 16) does not show any symmetric pattern. The only conclusion that we can find is that the 3rd and 4th pairs show somewhat symmetric pattern. E.coli o157 and salmonella did not show symmetry neither.

7. Concluding Remarks

Palindromes that appear in a sequence have biophysical meanings and form hairpin structures [13]. We proposed a palindrome searching algorithm (PA) with time complexity of O(N), where N is the size of a DNA sequence. We tested with E.coli K12. The longest palindrome was 1,456, which is unusual in random sequences. There were also many other long palindromes. By composing multiple palindromic pairs, we could generate multi-pair palindromic patterns - that is more sophisticated structure than hairpin. We only counted 3-pair palindromic patterns with E.coli K12 in this paper. Unfortunately we could not find any meaningful patterns from the results but hope the result would help biologists understand the sequence. In addition, we picked 30 long palindromes, and presented them as a type of 30-pair palindromic patterns. We can identify one type out of many possible 30-pair palindromic patterns - we call this type as "structural representation". The structural representations from different DNA sequences (E.coli K12, E.coli o157, and salmonella) formed different structures. Some researches showed symmetry of a genome [6], but when we plotted several longest palindromes of E.coli K12, E.coli o157, and salmonella, they did not show this symmetric shape.

Our algorithm is not a complete algorithm; therefore, in some very artificial sequences, even though palindromic pairs exist our algorithm could not detect them. For example, suppose a sequence is "atatatat···atatat" (size is S) it consists of only two complement characters "a" and "t". If we do not allow complement gap, the size of the palindromic pair should be the half of the sequence (S/2). But our PA may not be able to find it. To compensate this weakness we proposed another algorithm briefly; therefore, implementing this new approach is one of our future work. Mismatches in subsequences are not considered. We only are interested in extracting palindromes allowing errors, which will be a future work.

An advantage of this "Break and Gather" approach is its easy adaptability to other pattern searching problems. This method can be used for finding direct repeats with O(N) time complexity, where N is the length of a sequence. Our algorithm can be modified to scan RNA and Protein by allowing other characters such as "u". The structural representation scheme for identifying a DNA sequence needs to be examined with multiple sequences. Comparing our results of E.coli K12 with other researches on this genome may provide clues to understand the structure of n-pair palindromic patterns.

References

[1] A. Apostolico, D. Breslauer, and Z. Galil, Parallel dtection of all palindromes in a string", Theoretical Computer Science, 141:1, pp.163-173, 1995.

[2] A.H.L. Porto and V.C. Barbosa, Finding approximate palindromes in strings, Pattern Recognition 35, pp.2581-1591, 2002.

[3] D. Breslauer and Z. Galil, Finding all periods and initial palindromes of a string in parallel, Algorithmica 14:4, pp.355-366, 1995.

[4] D. Gusfield, Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology, Cambridge University Press, New York, NY, 1997.

[5] Gonick, Larry, and M. Wheelis. The Cartoon Guide To Genetics. New York: Barnes & Noble, pp.147-148, 1983.

[6] J.A. Eisen, J.F. Heidelberg, O. White, and S.L. Salzberg, Evidence for symmetric chromosomal inversions around the replication origin in bacteria, Genome Biology 1(6), 2000.

[7] J. Jurka, Origin and evaluation of alu repetitive elements,

in R.J. Maraia (Ed.), The Impact of Short Interspersed Element (SINEs) on the Host Genome, R.G. Landes, New York, NY, pp.25-41, 1995.

[8] J.T.L. Wang, Discovering active motifs in sets of related protein sequences and using them for classification, Nucleic Acids Research 22(14), pp.2769-2775, 1994.

[9] K. Shishido, N. Komiyama, and S. Ikawa, Increased production of a knotted form of plasmid pbr322 DNA in Escherichia coli DNA topisomeraes mutants, Journal of Molecular Biology, pp.215-218, 2003.

[10] L.J. Jensen, C. Friis, and D.W. Ussery, Three view of microbial genomes, Res. Microbiol 150, pp.773-777, 1999.

[11] National Center for Biotechnology Information, The complete genome sequence of Salmonella enterica serovar typhimurium LT2, http://www.ncbi. nlm.nih.gov/entrez/query. fcgi?cmd= Retrieve&db =nucleotide&list_uids=16763390&dopt =GenBank, 2003.

[12] National Center for Biotechnology Information, The complete genome sequence of Escherichia coli O157, 2001, http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?cmd=Retrieve&db=nucleotide&list_uids=16445223&dopt=GenBank, 2003.

[13] National Center for Biotechnology Information, The complete genome sequence of Escherichia coli K12, 1997, http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?cmd=Retrieve&db=nucleotide&list_uids=16127994&dopt=GenBank, 23 Apr. 2001.

[14] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman, Basic locak alignment search tool, J. Mol. Biol. 215, pp.403-410, 1990.

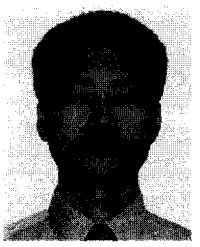
[15] S.F. Altschul also claimed that BLAST will have $O(N^2)$ time complexity to find all palindromes, Personal email communication, July 29, 2003.

[16] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. Introduction to Algorithms. New York: McGraw-Hill, pp.857-861, 1998.

[17] T.L. Bailey, Discovering motifs in DNA and protein sequences, University of California at Sandiego (PhD dissertation), 1995.

[18] T. Tsunoda, M. Fukagawa, and T. Takagi, Time and memory efficient algorithm for extracting palindromic and repetitive subsequences in nucleic acid sequences, Pacific Symposium on Biocomputing 4, pp.202-213, 1999.

[19] X. Guan and E.C. Uberbacher, A fast look-up algorithm for detecting repetitive DNA sequences, Pacific Symposium on Biocomputing, Singapore, pp.718-719, 1996.



김 형 래

e-mail : goddoes8@gmail.com

1997년 관동대학교 전자계산 공학과 (학사)

1999년 한국외국어대학교 경영정보대학원 (MBA)

2001년 Florida Institute of Technology, Computer Science(전산석사)

2005년 Florida Institute of Technology, Computer Science(전산박사)

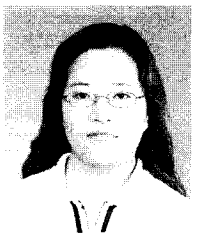
2001년 1월~2003년 5월 Research Assistant, Florida Institute of Technology

2003년 9월~2004년 5월 Teaching Assistant, Florida Institute of Technology, Computer Science

2004년 8월~2006년 5월 Assistant Professor, Livingstone College, Computer Information Systems

2006년 8월~present Associate Research Fellow Korea Employment Information Service, Information planning Team

관심분야 : Job Information Systems, Information Retrieval, Search Engine, Machine Learning,Web Personalization 등



정 경 희

e-mail : sept21@nate.com

1998년 관동대학교 전자계산공학과 (학사)

2000년 관동대학교 일반대학원 전자계산공학과(공학석사)

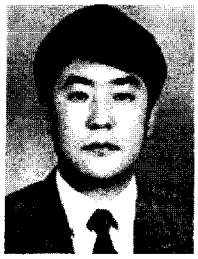
2002년~2004년 관동대학교 컴퓨터공학과 강사

2004년~2006년 세명대학교 정보보호학과 강사

2007년~현재 SK C&C Industry 사업2팀 재직

2003년~현재 관동대학교 전자계산공학과 박사과정

관심분야 : 고용정보시스템, 컴퓨터 그래픽스 등



전 도 홍

e-mail : dhjeon@kd.ac.kr

1985년 미국 Olahoma City University 컴퓨터과학(학사)

1987년 미국 Florida Institute of Technology 컴퓨터과학(석사)

1990년 미국 Florida Institute of Technology 컴퓨터교육학(박사)

1996년~2001년 관동대학교 전자계산소장, 중앙도서관장

1996년~2000년 한국 컴퓨터교육학회 이사

2000년~2002년 한국 컴퓨터교육학회 부회장

2000년~2004년 전국대학 정보전산기구 협의회 이사

1998년~현재 강원도 정보통신 상임 자문위원

1996년~현재 관동대학교 컴퓨터학과 교수

관심분야 : 컴퓨터그래픽스, neural networks, 패턴인식,등