A New Approach to Motif Templates Analysis via Compilation Technique

Dr. Hussein K. Al-Khafaji

dr.hkm1811@yahoo.com
Al-Rafidain University Collage

Ghada E. Kassim

ghadalove81@yahoo.com
Iraqi Commission for Computers and Informatics

Abstract: *Motif template assertion and analysis is compulsory operation in most of bioinformatics systems such as motif search, sequential pattern miner, and bioinformatics databases analysis. The motif template can be in any length, therefore, the typing errors increased according to the length of motif. Also, when the structure motifs are submitted to bioinformatics systems they require specification of their components, i.e. the simple motifs, gaps, and the limits of the gaps. This research proposed a context free grammar, GFC, to describe the motif structure, and then this CFG is utilized to design an interpreter to detect, debug the errors, and analyze the motif template to its* *components. All the errors of 100 motifs of length arranged from 100 Base to 10 KBase are detected. These motifs are entered by 10 data entries. The experiments showed high correlation between number of errors and number of gaps, size of simple motifs, and motif template size. The target code of the interpreter is the components of a submitted motif template to be used in bioinformatics systems as next steps.*

Keywords: Bioinformatics, DNA, Motif Template Analysis, DNA/RNA and Protein Sequential Pattern Mining.

1. Introduction

Bioinformatics is a branch of biological science which deals with the study of methods for storing, retrieving and analyzing biological data, such as nucleic acid (DNA/RNA) and protein sequence, structure, function, pathways and genetic interactions. It generates new knowledge that is useful in such fields as drug design and development of new software tools to create that knowledge. Bioinformatics also deals with algorithms, databases and information systems, web technologies, artificial intelligence and soft computing, information and computation theory, structural biology, software engineering, data mining, image processing, modeling and simulation, discrete mathematics, control and system theory, circuit theory, and statistics [1]. Bioinformatics derives knowledge by computer analysis of biological and molecular biology data. It is a rapidly growing branch of biology, highly interdisciplinary, and it uses techniques and concepts from informatics, statistics, mathematics, chemistry, biochemistry, genetics, physics, linguistics and other fields. The biological data can be the information stored in the DNA sequences, experimental results from various sources, three-dimensional protein structures, gene expression arrays, patient statistics, scientific literature, etc. An important part of research in bioinformatics is the development of methods for storage, retrieval and analysis of these data. DNA can be analyzed both as a text and as a molecule that interacts with a variety of other molecules [2]. The primary goal of bioinformatics is to increase the understanding of biological processes. The discriminating factor that differentiates bioinformatics from other approaches, however, is its focus on developing and applying computationally intensive techniques to achieve this goal. Examples include: pattern recognition, data mining, machine learning algorithms, and visualization. Major research efforts in the field include sequence alignment, gene finding, genome assembly, drug design, drug discovery, protein structure alignment, protein structure prediction, prediction of gene expression and protein–protein interactions, genome-wide association studies and the modeling of evolution [3].

2. Deoxyribonucleic Acid (DNA)

DNA is the hereditary material in humans and almost all other organisms. Nearly every cell in a person’s body has the same DNA. Most DNA is located in the cell nucleus (where it is called nuclear DNA), but a small amount of DNA can also be found in the mitochondria (where it is called mitochondrial DNA or mtDNA. The information in DNA is stored as a code made up of four chemical bases: adenine (A), guanine (G), cytosine (C), and thymine (T). Human DNA consists of about 3 billion bases, and more than 99 percent of those bases are the same in all people. The order, or sequence, of these bases determines the information available for building and maintaining an organism, similar to the way in which letters of the alphabet appear in a certain order to form words and sentences. DNA bases pair up with each other, A with T and C with G, to form units called base pairs. Each base is also attached to a sugar molecule and a phosphate molecule. Together, a base, sugar, and phosphate are called a nucleotide. Figure (1) shows the structure of DNA base pair. Nucleotides are arranged in two long strands that form a spiral called a double helix. The structure of the double helix is somewhat like a ladder, with the base pairs forming the ladder's rungs and the sugar and phosphate molecules forming the vertical sidepieces of the ladder .An important property of DNA is that it can replicate, or make copies of itself. Each strand of DNA in the double helix can serve as a pattern for duplicating the sequence of bases. This is critical when cells divide because each new cell needs to have an exact copy of the DNA present in the old cell [4]. DNA plays a fundamental role in the different bio-chemical processes of living organisms in two respects [5]:

- Firstly, it contains the information the cell requires to synthesize protein and to replicate itself. To be short, it is the storage repository for the information that is required for any cell to function.

- Secondly, it acts as a medium for transmitting the hereditary information (namely the synthesis plans for proteins) from generation to generation.

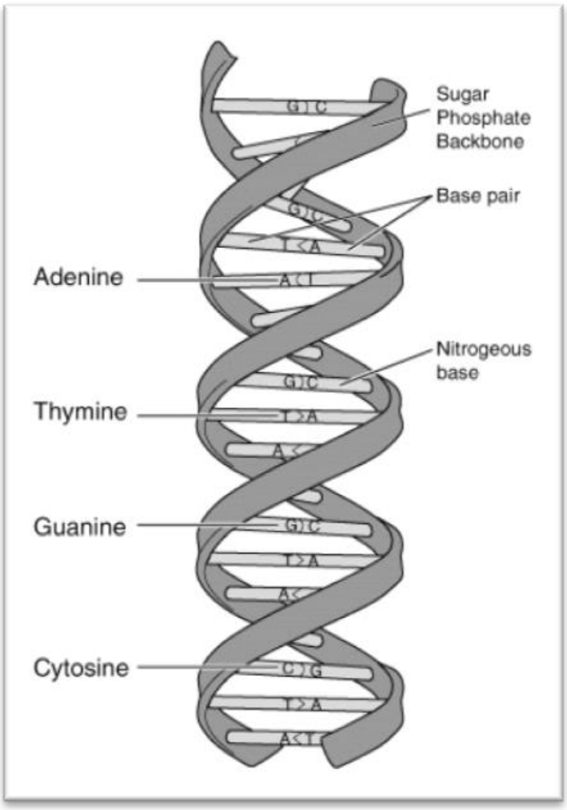

Figure (1) DNA is a double helix formed by base pairs attached to a sugar-phosphate backbone.

3. DNA Motif

Searching biological sequence(s) for motifs is a fundamental task in bioinformatics. Motifs can be represented as either patterns over a specific alphabet, or profiles (also called positional weight matrix (PWM)), which give the probability of observing each symbol in each position [6]. Generally, a DNA motif is defined as a nucleic acid sequence pattern that has some biological significance such as being DNA binding sites for a regulatory protein, i.e., a transcription factor. DNA motifs are often associated with structural motifs (will be illustrated later) found in proteins. Motifs can occur on both strands of DNA. Transcription factors indeed bind directly on the double-stranded DNA. Sequences could have zero, one, or multiple copies of a motif [7]. Motifs could be monad or composite; a pattern with single component is also called single pattern, most approaches to pattern discovery focus on monad patterns that correspond to relatively short contiguous string. But in the search for more complex methods have focus on composite motifs [8]. Composite motif can be thought of as “compound pattern” made of a list of monad motifs, or patterns, and a list of intervals that specify at what distances adjacent motifs should occur. It can be classified into two main types [9]:

1. If no variable gaps are allowed in the motif; it is called a *simple motif.*
2. If variable gaps are allowed in a motif; it is called a *structure motif.*

4. Motif Template

Motif template is a serious of sequential characters containing gap(s). This motif is here DNA motif reveals:-

- Simple motifs alphabet either in DNA alphabet or in International Union of Pure and Applied Chemistry (IUPAC) format.
- Motifs length, type and number of symbols in each simple motif.
- Motifs components, number of simple motifs.
- Gaps length, minimum and maximum gaps.

Structure motif is formally represented as:
M_1 {[l_1, u_1] M_2 {[l_2, u_2] M_3 }{….. M_n [l_n, u_n]} M_{n+1}}, where the structures surrounded by { and } are optional. So, for example, the following

GGGTGGGAAGGTCGT [5, 17] TTAGCGGGTAT

is a structure motif template which consists of two simple motifs in DNA alphabets, M1 [5,17] M2 such that M1 is GGGTGGGAAGGTCGT with length of 15 bases and M2 is TTAGCGGGTAT with length of 11 bases and variable gap between these two simple motifs as minimum gap of 5 bases and maximum gap of 17 bases. Accordingly, after 5 to 17 of don't care bases, the searcher has to look for M2. It is considered as an event and occurrence of the pattern, and when the number of occurrences equals or greater than minimum threshold it is regarded as frequent pattern [10].

From the above introduction, one can see that a motif is a sequence of characters arranged according to specific grammar. A motif may be of millions of bases, therefore, writing a motif template, for any purpose in bioinformatics' systems, suffers from syntax and semantic errors. Detection and correction of such errors are very difficult and effort and time consuming operation. This research produces a new approach to detect and correct the errors occurred in motif templates submitted to sequential pattern miner or any bioinformatics system. This approach depends on the principles of compiler design. The next section illustrates the design and implementation of the motif template interpreter.

5. Design of the Proposed Motif Template Interpreter

Because the motif template must have fixed structure, it is possible to automate the checking of the structure to meet its writing rules. Therefore, in this research an interpreter has been suggested to check the syntax and semantic structure of a motif template. It produces error messages when it finds any mistake(s) when the interpreter scans the motif template. The interpreter has been called DNA *Motif Template Interpreter, DNAMTI*. The design and implementation of DNAMTI is described in the figure (2). Its input is the motif template which is either from the samples has been saved or by writing it manually by a text sensitive editor. The output of DNAMTI is the components of the asserted motif template. It is similar to any translator, so, it consists of lexical analyzer, syntax analyzer, semantic analyzer, and code-generator-like phase.

5.1 Lexical Analyzer

Lexical analyzer or scanner is the process where the stream of characters making up the motif template is read from left-to-right and grouped into tokens. Tokens are sequences of characters with a collective meaning. The lexical analyzer takes a source motif template as input, and produces a stream of tokens as output. The lexical analyzer might recognize particular instances of tokens such as:-

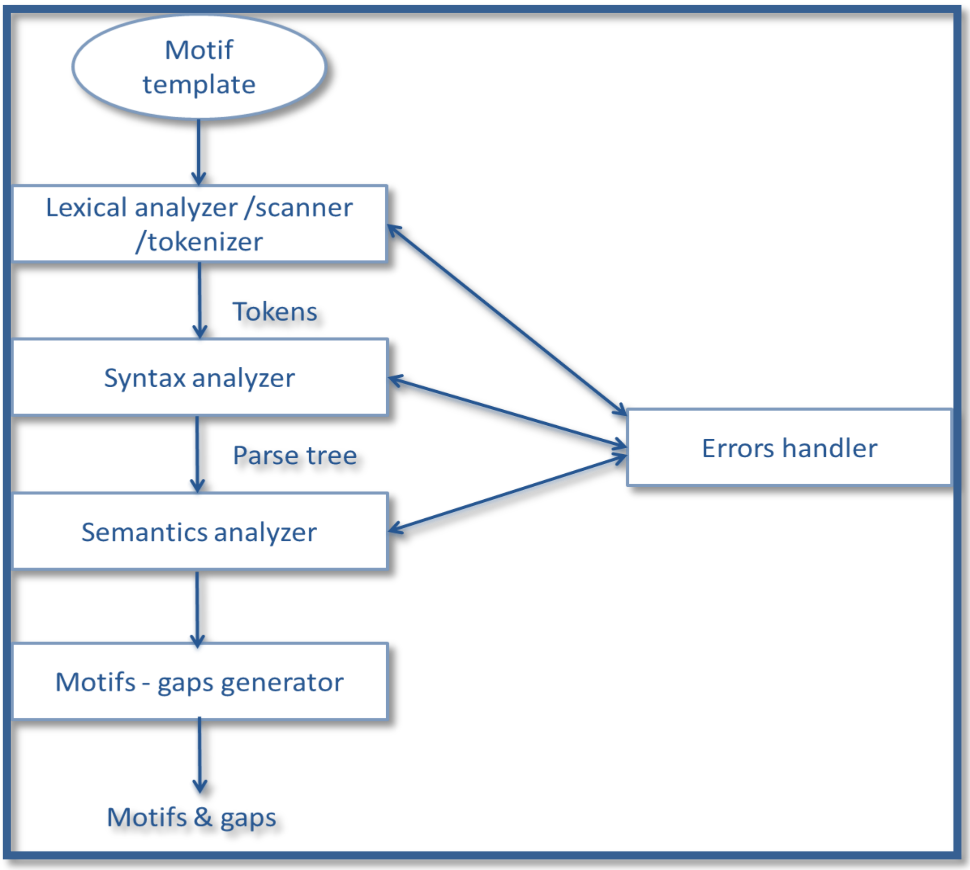

Figure (2) Phases of the DNAMTI

- Numbers for gaps limits.
- String constant token for simple motifs such as AC, CC, TCTT, etc.
- Reserved characters such as "[",']",",", etc.

Such specific instances are called lexemes. A lexeme is the actual character sequence forming a token; the token is the general class that a lexeme belongs to. The scanner is tasked with determining that the input motif template can be divided into valid tokens in the source motif template, but has no smarts about which token should come where. Few errors can be detected at the lexical level alone because the scanner has a much localized view of the motif template without any context. The scanner can report about characters that are not valid tokens (e.g., illegal or unrecognized characters such as the characters which are not belonging to DNA alphabet). It does not look for or detect garbled sequences, tokens out of place and undeclared sequences characters. For example, the following input will not generate any errors in the lexical analysis phase, because the scanner has no concept of the appropriate arrangement of tokens for a motif template

AATACAT] 4 [20 CCCTAGATACA,

The syntax analyzer will catch this error later in the next phase.

Furthermore, the scanner has no idea how tokens are grouped. In the above sequence, it returns, as shown in table (1), the following tokens values and types (AATACAT, simple motif), (], RB), (4, NUM), ([, LB), (20, NUM) and (CCCTAGATACA, simple motif) as six separated tokens, having no idea they collectively in right form or not, where RB, LB, and NUM are Right Bracket, Left Bracket, and number respectively. A fragment of the pseudo code of this phase is shown in the figure (3). The returned values from the scanner will be passing on to the syntax analyzer in the next phase. Some of these tokens types are:-

- Motif tokens: one or more sequences of DNA bases.
- Left and right square brackets [and].
- Integers which represent the minimum and maximum values of the gaps.
- Reserved character such as comma, =, etc.

Table (1) The Output Tokens From Lexical Analyzer Phase.

Tokens values	Tokens types
AATACAT	Simple motif
[	Left curly bracket
4	Minimum gap
,	Comma
20	Maximum gap
]	Right curly bracket
CCCTAGATACA	Simple motif

As shown in the figure (3), the lexical analyzer is implemented as a routine which is frequently called by the syntax analyzer to produce a token value and token type. It consumes the motif template character by character until the end of the template. The name of this routine is tokenizer in the algorithm presented in figure (3). This algorithm produces one token at each call. Formally, a token consists of the token value and its type, consider table (1).

```
Tokenizer algorithm

Input: motif template

Output: (Token_ value, Token type)

1.  Token_ value ="";
2.  Ch=getchar (motif template); // separate a characters from motif template
    and store it in Ch
3.  Case   Ch  o f
4.     {
5.       ‘A’,’C’, ‘G’ ,’T’ :
6.        {
7.           While Ch  in ['A’,’C’,’G’,’T’]    DO
8.            {
9.              Token_value =Token_value+ch;
10.            Ch=getchar (motif template);
11.           }
12.       Concat(motif template,Ch); //concatenate motif template and Ch//
13.       return (("simple motif",Token_value));
14.     }
15.    ‘[‘  : { Token_value  =”[“ ; return ((“[“,“LB”))}
16.    ‘]’  : { Token_value  =”]” ; return ((“]”,“RB”))}
17.    ‘,’  : { Token_value  = “,” ; return ((“,”,“comma”)};
18.    ‘0’| ‘1’ | ‘2’| ‘3’ | ‘4’ |’5’ | ‘6’ | ‘7’ | ‘8’ | ‘9’ : ….
```

Figure (3) Fragment of Lexical Analyzer's pseudo code.

```
    {
      While  Ch  in  ['0','2','3','4','5','6','7','8','9']  do
       {
           Token_vlaue = Token_value +Ch;
           Ch = getchar(motif template);
       }
      Concat (motif template, ch);
      return (("NUM", Token_value));
    }
    error (" illegal character ");
Call motif template editor;
Exit;
}   // end case
```

Figure (3) Cont.

5.2 Syntax Analyzer

Where the lexical analyzer splits the input motif template into tokens, the purpose of syntax analyzer is to recombine these tokens. Not back into a list of characters, but into something that reflects the structure of the motif template. This "something" is typically a data structure called the syntax tree of the motif template. As the name indicates, this is a tree structure. The leaves of this tree are the tokens found by the lexical analyzer phase, and if the leaves are read from left to right, the sequence is the same as in the input motif template. Hence, the important duty of the syntax tree is describing how these leaves are combined to form the structure of the tree and describing how the interior nodes of the tree are labeled. The syntax of motif template was described by a context free grammar. Part of the rules of the grammar of the template is shown in figure (4).

```
Rule 1 :  MT → Motif
Rule 2 :  MT→Motif GAP MT
Rule 3 :  Motif →a|c|g|t
Rule 4 :  Motif → a Motif
Rule 5 :  Motif → c Motif
Rule 6 :  Motif → g Motif
Rule 7 :  Motif → t Motif
Rule 8 : GAP→ [NUM,NUM]
Rule 9 : NUM→digit NUM
Rule 10 : NUM→ digit
Rule 11 : digit →0|1|2|3|4|5|6|7|8|9
…
```

Figure (4) Part of motif template's CFG rules

A fragment of the syntax analyzer's pseudo code is shown in the figure (5).

Syntax analyzer algorithm

Input: tokens stream

Output: parse tree // its contents depends on the applied rules of the CFG//

```
1.  While there are more tokens
2.  {   Tokenizer; // call the lexical analyzer//
3.      If  token type $\neq$ "simple motif"
4.        {  Error (" simple motif expected ");
5.            Call motif template editor ;
6.           Exit ;  }
7.      Else {Construct parse tree (token value, token type );
8.            Tokenizer ;
9.            If token type   $\neq$ "LB"
10.              { Error ("LB expected ");
11.                Call motif template editor; exit; }
12.           Else  { Tokenizer;
13.               Construct parse tree (token value, token type );
14.                 If token type  $\neq$ " NUM"
15.                    { Error (" minimum gap expected");
16.                        Call motif template editor ; exit}
17.                 Else { Tokenizer ;
18.                         Construct parse tree (token value, token type );
19.                         If token type  $\neq$ " comma"
```

Figure (5) Fragment of syntax analyzer pseudo code

```
20.       { Error(" comma expected");
21.          Call motif template editor ; exit }
22.       Else { Tokenizer;
23.              Construct parse tree (token value, token type );
24.              If token type $\neq$ "NUM"
25.             { Error ("maximum gap expected");
26.              Call motif template editor; exit }
27.               Else { tokenizer;
28.                  Construct parse tree (token value, token type );
29.                  If token type $\neq$ "RB"
30.                  { Error(" RB expected");
31.                 Call motif template editor; exit; }
32.                   Else { tokenizer;
33.                          Construct parse tree (token value, token type );
34.                          If token type $\neq$ "simple motif "
35.                         { error ("simple motif expected");
36.                          Call motif template editor ; exit;}
    }}}}}}…
37. } //end while//
```

Figure (5) Cont.

The alphabet of the DNA motif template grammar is {a, c, g, t, , , [,], 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, =}. Therefore, if the submitted example is AC[5, 7]GT, the parse tree is as shown in figure (6).

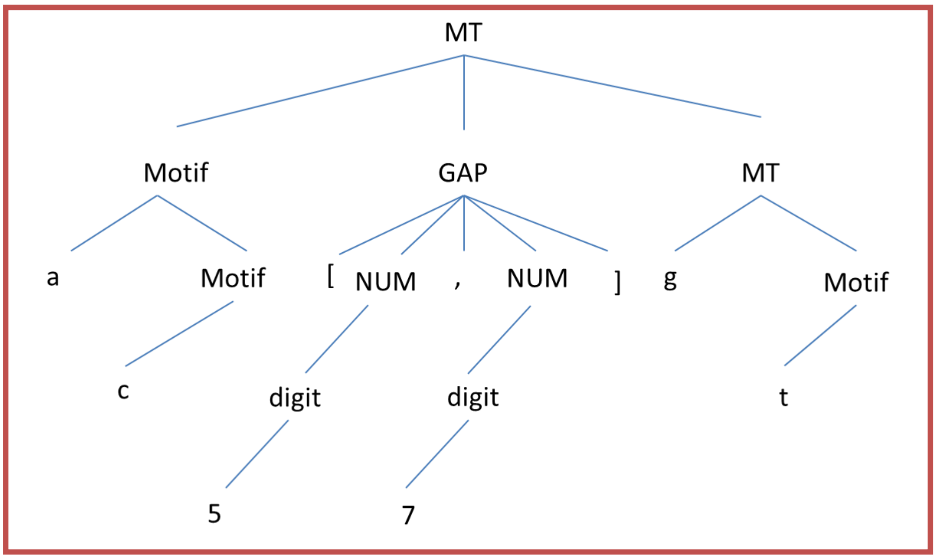

Figure (6) Motif template parse tree

Accordingly, if the motif template, AC [5, 7] GT is taken, then the validation process will be as follows:

MT →Motif GAP MT ………………………….….. ..Rule 2

MT →a Motif GAP MT …………………………. ...Rule 4

MT → ac GAP MT ……………………........................Rule 3

MT → ac [NUM,NUM] MT …………………………Rule 8

MT → ac [digit, NUM] MT……………………………Rule 10

MT →ac [5,NUM] MT……………………………....Rule 11

MT →ac [5,digit] MT…………………………………..Rule 10

MT →ac [5,7] MT………………………………………Rule 11

MT →ac[5,7] Motif ……………………………………...Rule 1

MT →ac[5,7] g Motif……………………………………Rule 6

MT →ac [5,7] gt ………………………………………Rule 3

5.3 Semantic Analyzer

It's well known how an input motif template is syntactically checked by the syntax analyzer; it should now know how to analyze the input motif semantically. In this phase the properties that cannot be captured by context free grammar are captured by semantic phase. From implementation view point, there is no clear cut between the syntax and semantic phase because some of semantic errors are manipulated by the syntax phase such as the values of gaps limits. For example the following template AACG [17, 5] TTG is syntactically correct but it is semantically erroneous template because 17 is greater than 5. Also, there is a semantic error related to negative number. Figure (7) depicts a fragment of pseudo code related to min and max gap.

```
Semantic analyzer algorithm

Input : parse tree

Output: semantically checked parse tree.// semantic error free parse tree

.
.
.
10.  // see MT nonterminal in figure 5 //
11. While (more MT nonterminal in parse tree) do
12. {
13.    get next MT nonterminal ;
14.    If MT.GAP  is not null
15.    If   MT.GAP.NUM1   >   MT.GAP.NUM2
16.      {
17.        error (" min gap should be less than max gap!");
18.        Call motif template editor;
 19.       Exit;
20       }
      .

      .
      .
100.  }
```

Figure (7) Fragment of semantic analyzer pseudo code.

5.4 Motifs and Gaps Generator

The motifs and gaps generator is the final activity of DNAMTI. This phase has two tables, the first is for simple motifs and the second is for minimum and maximum gaps. The algorithm takes the tokens from the semantic analyzer phase one after another. It checks them if they are simple motif, the generator saves it in simple motifs table and if it is a gap, it will be saved in gaps table. At last convert the minimum gap and maximum gap to number. Figure (8) shows the algorithm for motifs and gaps generator.

```
Motifs & gaps generator algorithm

Input: semantically checked parse tree.

Output: simple motifs table, gaps table

1. Motif_counter=Gap_counter=1;
2. While more MT nonterminal in the parse tree
3. {
4.   Simple_motif[Motif_counter ++];
5.   Get_next_motif ( parse_tree);
6.    GAP[GAP_counter]. Min=get_next-GAP_number(parse_tree);
7.    GAP[GAP_counter ++] . Max = get_next_GAP-number(parse_tree);
8. }
```

Figure (8) Fragment of Motifs and Gaps Generator Pseudo Code

For example, the following structure motif: ACCC [5, 7] TCG [3, 10] CACCGT will be converted to GAP and simple motif tables as presented in figure (9). The get-next-motif is a routine which specifies the next motif in the parse tree according to the grammars illustrated in figure (5). Also, get-next GAP-number specifies the next number in a GAP of parse tree. The output from the interpreter stage is error free motif template plus two tables, simple motifs table and gaps table. Only the error free template will be used in the next module.

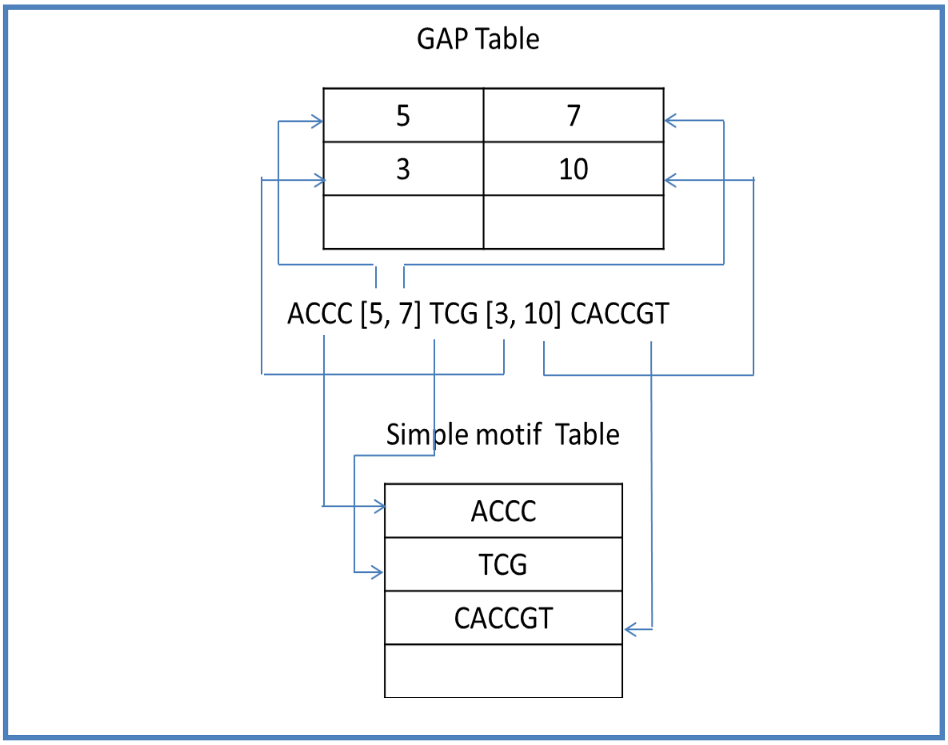

Figure (9) Simple motifs and gaps tables

6. Experiments Results

The idea of this research is born from the difficulties which are faced when we submit a motif template for bioinformatics sequential pattern miner. To study the problem, a sample of 100 motif templates is selected and arranged from 100 bases to 10 Kbase. The number of gaps in these templates are of sized arranged from 5 to 1000 gaps. These samples are distributed on 10 data entries. The average number of errors occurred by each of them are 813 errors, i.e. the total number of errors is 8130. The goal of the experiments is to prove the efficiency of the motif interpreter in addition to determining the relation between some factors and number of errors such as the size of the motif template, the number of gaps in a template and the lengths of simple motifs constructing the template. Table (2), Table (3), Table (4), and Table (5) depict the relations of the number of errors related to the motif template size, number of gaps, and simple motif length respectively.

Table (2) Relation of motif template size and number of errors

Motif Template Size + 70 characters of 10 gaps	Average Number of Errors for 10 Data Entries of 10 Size Categories
1 Kbase	11
2 Kbase	19
3 Kbase	22
4 Kbase	35
5 Kbase	46
6 Kbase	70
7 Kbase	91
8 Kbase	140
9 Kbase	175
10 Kbase	204
	813 errors

The experiments showed that the interpreter has the ability to detect all the errors successfully. The motif templates in the experiments are measured in Kbase. In these experiments, all the gaps have the general form [DD, DD] where D is a digit, i.e., each gap is represented by seven characters; [, d, d, ,, d, d, and]. This assumption is adopted to avoid the variation of gap length which may affect the results.

Table (3) Relation of number of gaps to the number of errors in a motif of size 1 KBase with constant length of simple motifs among the gaps

No. of gaps	Simple Motif Length	No. of Errors
10	102	117
20	51	101
30	34	94
40	25	82
50	20	73
60	17	76
70	14	92
80	12	97
90	11	113
100	10	118

Table (4) Relation of number of gaps to the number of errors in a motif template of size 1 KBase with variant length of simple motifs among the gaps

No. of gaps	No. of Errors
10	31
20	32
30	66
40	86
50	100
60	147
70	153
80	191
90	208
100	224

Table (5) Relation of simple motifs length to the number of errors in a motif template with 10 gaps and variant sizes

Simple Motif Length	Motif Template Size (Base)+ No. of characters in the Gaps	No. of Errors
10	110+70=180	12
20	220+140=360	17
30	330+210=540	26
40	440+280=720	37
50	550+350=900	69
60	660+420=1080	91
70	770+490=1260	93
80	880+ 560=1440	114
90	990+630=1620	147
100	1100+700=1800	166

7. Conclusions

The experiments results showed that the interpreter has the ability to detect all the errors successfully. The number of errors in the experiments and the ability of the interpreter to detect these errors elucidated the necessity of designing such interpreter and proved the correctness of utilization of the compiler design principles in detecting, correcting and analyzing the motif templates.

The experiments presented unexpected results such as:

1) High correlation between the number of errors and number of gaps in a motif, recall table (3) and table (4).
2) High correlation between the number of errors and the lengths of simple motifs constructing the motif template, recall table (5).

8. Suggestions for Future Work

Many developments for DNAMTI can be done in the future such as:

1. Development of the DNAMTI to handle the RNA, IUPAC, and protein templates.
2. Development of the DNAMTI to be a complete tool to convert among DNA, RNA, protein, and IUPAC format.
3. The experiments showed that the data entries produce undetectable base transposition errors. The detection requires providing DNA Meta rules to describe the nature of the DNA and the relations among bases. A recommended future development is providing the ability of joining such meta rules about the DNA underhand with DNAMTI.

References

1. Paulien Hogeweg, "The Roots of Bioinformatics in Theoretical Biology", Springer, 2011.
2. Frederick B. Marcus, "Bioinformatics and Systems Biology", Springer, 2008.
3. Attwood T. K., Gisel A., Eriksson N-E., and Bongcam-Rudloff E. "Concepts, Historical Milestones and the Central Place of Bioinformatics in Modern Biology: An European Perspective", Bioinformatics-Trends and Methodologies, 2012

4. Genetics Home Reference (http://ghr.nlm.nih.gov/), "Cells and DNA", National Center for Biomedical Communications, 2012.
5. A. M. Carvalho and A. T. Freitas, "An Efficient Algorithm for the Identification of Structured Motifs in DNA Promoter Sequences", IEEEACM Transcriptions on Computational Biology and Bioinformatics, 2006.
6. Yongqiang Zhang and Mohammed J. Zaki, "SMOTIF: Efficient Structured Pattern and Profile Motif Search", BioMed Central Ltd, 2006.
7. Modan K Das and Ho-Kwok Dai, "A Survey of DNA Motif Finding Algorithms", Fourth Annual MCBIOS Conference, Computational
Frontiers in Biomedicine, 2007.
8. C. de Boer and T. Hughes, "YeTFaSCo: A Database of Evaluated Yeast Transcription Factor Sequence Specificities", Nucleic Acids Research, 2012.
9. L. Yan and Z. Sun, "Closed Structured Patterns and Motifs Mining Without Candidate Maintenance", Second International Conference on Future Information Technology and Management Engineering, 2009.
10. A. Brazma and I. Jonassen, "Approaches to the Automatic Discovery Of Patterns in Bio Sequences", Journal of computational biology and Computational Molecular Cell Biology, 2009.

أتجاه جديد لتحليل نماذج الحمض النووي باستخدام تقنية الترجمة

أ.م.د. حسين كيطان الخفاجي
dr.hkm1811@yahoo.com
كلية الرافدين الجامعة

م.م. غادة عماد قاسم
ghadalove81@yahoo.com
الهيئة العراقية للحاسبات والمعلوماتية

المستخلص

ان عملية ادخال وتحليل نموذج الحمض النووي هي عملية اجبارية مطلوبة في اغلب الانظمة الاحيامعلوماتية، مثل البحث عن نمط ما، تعدين النماذج الاحيائية المتسلسلة، وتحليل قواعد البيانات الاحيائية. ان نموذج الحمض يمكن ان يكون باي طول وعليه فان اخطاء الطباعة تزداد مع طوله طرديا، فضلا عن ذلك ان الانظمة الاحيامعلوماتية تحتاج الى تحليل النموذج الى مكوناته مثل نماذجه البسيطة، فضاءاته، والحدود الدنيا والعظمى لتلك الفضاءات.

هذا البحث يقترح قواعد محررة المحتوى لوصف معمارية النموذج الحمضي ثم استخدامه لتصميم مفسر لاكتشاف الاخطاء القواعدية والدلالية وتصحيحها ان وجدت ومن ثم يحلل النموذج الى مكوناته المذكورة. كل الاخطاء قد اكتشفت في 100 عينة من النماذج الحمضية والتي تتراوح اطوالها بين 100 قاعدة الى 10 كيلوقاعدة والمدخلة من قبل 10 من مدخلي البيانات. التجارب اوضحت ان هنالك ارتباطا وثيقا بين عدد الاخطاء وبين حجم النموذج الحمضي، حجم النماذج البسيطة المكونة له، وعدد فضاءاته. وكل النماذج المدخلة قد حللت وان الشفرة الهدفية الناتجة من المفسر

هي مكونات النموذج الحمضي والتي ستستخدم في الانظمة الاحيامعلوماتية كخطوة لاحقة.

الكلمات الرئيسية: الأحيامعلوماتية، تعدين قواعد بيانات الحمض النووي، تحليل مقاطع الحمض النووي، تعدين النماذج المتسلسلة في الحمض النووي والبروتين.