A New Model for Finding Approximate Tandem Repeats in DNA Sequences

Qingshan Jiang*
Shenzhen Institute of Advanced Technology, Chinese Academy of Science, Shenzhen, China
Software School, Xiamen University, Xiamen, China
Email: qjiang@xmu.edu.cn

Sheng Li
Software School, Xiamen University, Xiamen, China
Email: gerry11@tom.com

Shun Guo
School of Information Science and Technology, Xiamen University, Xiamen, China
Email: gsgowell@gmail.com

Dan Wei
Cognitive Science Department, Xiamen University, Xiamen, China
Fujian Key Laboratory of the Brain-like Intelligent Systems (Xiamen University), Xiamen, China
Email: danweiwd@yahoo.cn

***Abstract*—In gene analysis, finding approximate tandem repeats in DNA sequence is an important issue. SUA_SATR is one of the latest methods for finding those repetitions, which suffers deficiencies of runtime cost and poor result quality. In order to detect approximate tandem repeats in genomic sequences more efficiently, we propose a new model based on a novel algorithm MSATR and an optimized algorithm *m*MSATR in this paper. The model uses the Motif-Divide method to improve the performance, which results in the proposal of algorithm MSATR. By introducing the definition of CASM to reduce the searching scope and optimizing the original mechanism adopted by MSATR, the *mMSATR* algorithm makes the detecting process more efficient and improves the result quality. The theoretical analysis and experiment results indicate that MSATR and *m*MSATR is able to get more results within less runtime. These algorithm*s* are superior to other methods in finding results, and it greatly reduces the runtime cost, which is of benefit when the gene data becomes larger.**

***Index Terms*- DNA sequence mining; approximate tandem repeat; motif-similarity**

I. INTRODUCTION

Bioinformatics defined as the application of computational techniques to understand and organize the information associated with biological macromolecules [1] is a discipline that combines Biology, Computer Science, Mathematics and knowledge in other realms[2][3][4]. DNA, the mystical sequence where life starts, has been one of the major research objects in Bioinformatics. In those DNA sequences disperse iterations of nucleotide motifs called Tandem Repeats (TRs). TRs' genetic and evolutionary mechanisms remain controversial; however it is believed that they are functionally important for gene transcription, translation, chromatin organization, recombination, DNA replication, cell cycle, etc[4][5][6][6][15]. Currently TRs, as important genetic makers, has been prevalently applied in realms such as paternity testing, forensic investigations and so on. Therefore, it is essential to find and study those TRs by using Data Mining technologies[2][15].

The TRs that current algorithms are trying to find include those that are totally similar to one another and those that are partially similar. An exact tandem repeat in a genomic sequence is a string of nucleotides that consists of multiple consecutive occurrences of a substring called a motif. For instance, AAATTAAATT is an exact tandem repeat of a motif, AAATT, of length 5. Algorithm [7][10][11][15] are aimed to detect the TRs that are completely similar. However, because of the mutation, migration, inversion of gene, those tandem repeats commonly are not completely similar. Thus, the concept Approximate Tandem Repeat (ATR) was brought up. It is defined as a string of nucleotides repeated consecutively at least twice with small differences between the instances. Finding ATRs in a sequence is a more complicated task than finding TRs and has been addressed by several papers during recent years[13][14][15][17]. One of them is algorithm SUA_SATR (Succeeding Unit Array_ Search segment-similarity based Approximate Tandem Repeats)[4] that uses Succeeding Unit Array (SUA) as the index structure and a new similarity measurement, and it is shown that SUA_SATR is superior to traditional algorithms in both runtime and result quality.

Based on the ground work of SUA_SATR, this paper put forward a new model, which leads to a novel algorithm MSATR (Motif-divide based Search Approximate Tandem Repeats) and an optimized

*Corresponding Author, email: qjiang@xmu.edu.cn.

doi:10.4304/jsw.6.3.386-394

algorithm *m*MSATR (modified MSATR). By introducing a new index structure based on Motif-Divide method, this model can achieve better efficiency both in runtime and result quality, which has been demonstrated by the fact that MSATR outweighs SUA_SATR in runtime efficiency as well as result quality. To further optimize the performance of MSATR, during dividing sequence into motifs, *m*MSATR implements some analysis for each motif to reduce the searching scope for later process, which results in less runtime. Besides, *m*MSATR made some effort to improve the result quality. Experiments have shown that *m*MSATR and MSATR costs much less runtime and detects more ATRs when compared with SUA_SATR. Meanwhile, *m*MSATR is superior to MSATR.

The remainder of this paper is organized as follows. Section II would introduce the related work and Section III then would elaborate the new model and algorithms MSATR and *m*MSATR. After that, Section IV would compare three algorithms by analyzing the experimental results and demonstrate the system based on the research. At last it is the conclusions by Section V.

II. RELATED WORK

Earlier algorithms use the *Edit Distance* as the measurement of similarity [13]. It suffers deficiency of low efficiency and it is only applied for finding short TRs in DNA sequence with a limited length. Later, Kurtz [14] etc. proposed the algorithm REPuter which is based on structure of suffix-tree, and its efficiency is improved; however, the result is not satisfactory. In 2007, algorithm SUA_SATR(Succeeding Unit Array_ Search segment-similarity based Approximate Tandem Repeats)[15] was proposed, which uses Succeeding Unit Array (SUA) as the index structure and Hamming Distance as the similarity measurement. Compared to the REPuter [14], SUA_SATR has a smaller space complexity, and it is easier and faster to build up the index structure. What is more, the similarity of SUA_SATR is more reasonable, and the efficiency is improved one step further. The model of finding ATRs is shown in Fig. 1.

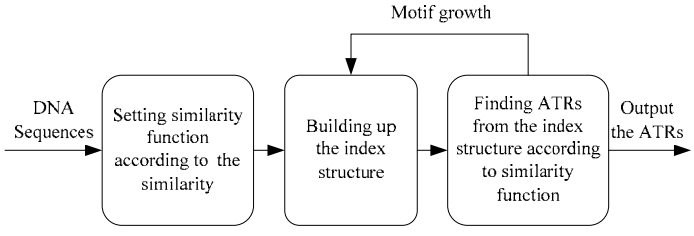

Figure 1. The model of finding ATRs

The steps of finding ATRs are as follows:

1. Defining different similarities of ATRs, such as Edit Distance and Hamming Distance etc;
2. Building up different index structure, for instance, the structure of suffix-tree and the succeeding array;
3. Scanning and Finding ATRs from the index structure according to similarity function;
4. Growing the motif;
5. Repeating from step 2 to 4 until no ATRs can be found.

Algorithm SUA_SATR [15] uses Succeeding Unit Array (SUA) as the index structure and Hamming Distance as the similarity measurement. SUA_SATR is superior to traditional algorithms in finding results and saving runtime.

Considering the influence of the gene lengths on similarity measurement, SUA_SATR algorithm introduces a new similarity measurement which is more reasonable. Furthermore, SUA_SATR proposes SUA as an index, and it reduces the times of irrelevant subsequence comparison thereby improving the time efficiency. In addition, SUA_SATR can mine more TRs which conform to the similarity measurement than traditional methods.

SUA_SATR applied SUA to find segment-similarity based approximate tandem repeats (ATR) in DNA sequence *seq*. SUA_SATR finds ATRs by the following steps:

1. Defining the similarity function based on Hamming Distance;
2. Dividing *seq* into motif units, putting them and their corresponding position relationships into an array, Sorting the array by the order of alphabet, and then the index structure of SUA is obtained;
3. Finding ATRs by traversing the index structure.
4. Growing the motif. Motif growth increases the length of each motif unit which is in SUA.
5. Repeating the above steps until no ATRs can be found.

During the process of finding SATRs from the index structure, comparisons between pairwise motif units are required.

The algorithm SUA_SATR is summarized as follows:

Algorithm SUA_SATR

Input: DNA sequence *seq*, similarity function *S*, similarity threshold *r*, minimum periods *p;*

output: SATRs

Begin

```
While (the motif in SUA can be grown) do
  for (each row in index structure) do
    If (the similarity between current row and the motif of
          existing similar segment ≥ r)
    {
      the depth of current similar segment ++;
      Signing the row as the ID of current similar
          segment;
    } else signing the row as a new ID
  for (each row which has no sign)
   Choosing the row of lowest starting position as current
      row;
   Successor = the succeeding motif of current row;
   While(Successor and current row are in the same
      segment)
   {
     SATR period ++;
     Signing current row;
     Successor = the succeeding motif;
      }
```

```
        end while
        if (successor meets the conditions of similarity)
          SATR period ++;
        if (SATR period >p)
          Output SATR;
        Growing motif;
       end for
     end while
     Output SATRs;
End
```

From the pseudo code of SUA_SATR, we can see that the core of SUA_SATR is building up the index structure of SUA. According to similarity function, SATRs can be formed by finding and joining the similar motif which is adjacent. However, attention should be paid to the following problems:

1. During the process of finding SATRs from the index structure, pairwise motif units need to be compared, even though they are nonadjacent. The efficiency of algorithm SUA_SATR remains to be improved.
2. During building the index structure of SUA, genomic sequences are divided according to the order of alphabet. Therefore, the motif units in the SUA of SUA_SATR must have the same first letter, which makes the motifs in each ATRs share the same attribute. Namely, the result it can find is limited.

In order to solve two above problems, we propose a new index structure which is based on the idea of Motif-Divide[2].

A. Related Concept

A DNA sequence is a sequence of symbols from the nucleotide alphabet Σ={A, C, G, T}. Tandem repeats exist in DNA sequences generally can be divided into two class, which are (exact) tandem repeats and approximate tandem repeats defined as follows:

Definition 1 **(Exact) Tandem Repeats**: An exact tandem repeat (TR) is a sequence that contains two or more contiguous copies of identical segments (referred as to motifs).

Copying errors happen in DNA sequences due to different external and internal factors, such as substitution, insertion, deletion, duplication, and contraction. Thus, the definition of approximate tandem repeats is given as;

Definition 2 **Approximate Tandem Repeats**: An approximate tandem repeat (ATR) is a sequence that contains two or more contiguous copies of similar segments.

Examples of TR and ATR are listed in **Error! Reference source not found.**.

TABLE I.
EXAMPLES OF TR AND ATR

Type	*Sequence*	*Motif*
TR	AGG AGG AGG AGG	AGG
ATR	AGG AGC AGG AGT	AGG

MSATR uses the same similarity measure as SUA_SATR [15] algorithm, the definitions are given as follows:

Definition 3 **Hamming Distance**: For two segments *X*, *Y* of length *n*, the distance between *X* and *Y*, $d_H(X, Y)$, is the number of sites where the corresponding nucleotides differ, or equivalently, the minimum number of substitutions required to convert *X* to *Y*.

Definition 4 **Motif-similarity**[2]: For two motifs X and Y of length *n*, the similarity between the two motifs is $d_H(X, Y) / n$.

And in order to find ATRs, the definitions are given as follows.

Definition 5 **MATR** (Motif-similarity based Approximate Tandem Repeats)[2]: For sequence $T = T_1T_2...T_p\ (p \geq 2)$, if the similarity of any two motifs T_i and T_j ($0 \leq i, j \leq p$) meets the similarity threshold *r*, then T is a MATR, and *p* is its periods.

For example, for a sequence *S*=ACCT|AGCT|AACT|ATCT, where *r*=0.75 and the length of motif is 4. The Motif-similarity of any two motifs is (4-1)/4=0.75, so the motifs in *S* form a MATR.

Definition 6 **Motif-Divide** Method [2]: For a sequence of length *n*, *seq*, it is divided *k* times according to *k*, which is the length of motif. Every division starts from the i^{th} $(0 \leq i \leq k-1)$ position and *seq* is divided into $\lfloor (n-i)/k \rfloor$ successive motifs. Then these motifs are put into arrays in turn. There will be *k* arrays after dividing. Obviously, the total number of elements in *k* arrays is less than *n*.

For example, for a sequence *S*=AGTTCTAACAGGAA GACGT, where *k*=4. According to *k*, S is divided into motifs of length 4. After putting these motifs into 4 arrays, the index structure consisting of 4 arrays is listed in TABLE II. .

TABLE II.
MOTIF-DIVIDE INDEX STRUCTURE OF *S*

Array	**Motifs**			
Array1:	AGTT	CTAA	CAGG	AAGA
Array2:	GTTC	TAAC	AGGA	AGAC
Array3:	TTCT	AACA	GGAA	GACG
Array4:	TCTA	ACAG	GAAG	ACGT

III. THE NEW MODEL FOR FINDING ATRS

To solve the problems mentioned above in SUA_SATR, we propose a new model of finding ATRs to improve the performance. The new model shows as Fig. 2.

The steps of the new model to find ATRs are as follows:

1. Defining different similarities of ATRs, such as Edit Distance and Hamming Distance etc;
2. According to the length of motif, sequence is divided into many motifs of same length;

3. Building up different index structure based on Motif-Divide;
4. Scanning and Finding ATRs from the index structure according to similarity function;
5. Growing the motif;
6. Repeating from step 2 to 5 until no ATRs can be found.

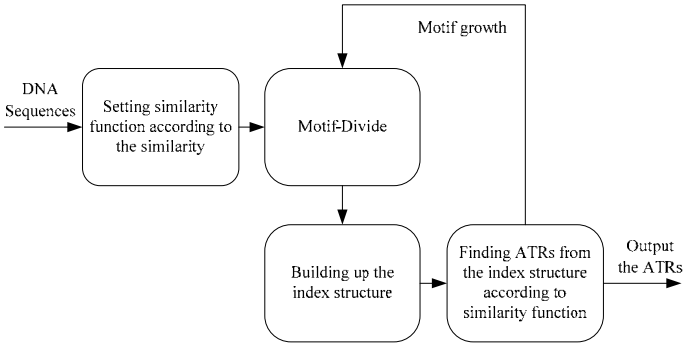

Figure 2. The new model of finding ATRs

Based on the new model of finding ATRs, we discuss on MSATR algorithm and MSATR's optimized algorithm mMSATR to improve the performance of SUA_SATR algorithm.

A. Algorithm MSATR

We introduce the idea of Motif-Divide to build up the array index in which each unit is a motif divided from the DNA sequence and adjacent to the one next to it. By checking the similarity of adjacent motifs and joining similar ones, algorithm is able to find the ATRs more efficiently. What is more, the ATRs found are not limited by the rule in SUA_SATR that the first letter must be the same.

In order to find all the MATRs in DNA sequence *seq*, MSATR consists of two steps, *dividing* and *joining* [2].

Dividing: First, *seq* should be divided into short motifs of length *k*. Because where the dividing starts matters, algorithm needs to do the dividing k times. Each time starts from position i ($0 \le i < k$), and the motifs divided each time are stored in a separate array. However, the end of the *seq* could be divided into a motif whose length is less than *k*. For those motifs, algorithm chooses to ignore them, instead of putting them in the array for further detecting.

Joining: Each array needs to be scanned, and MSATR calculates the similarity among motifs to see whether they should be joined.

Given the DNA sequence *seq*, the length scope (*a*, *b*) of ATRs that we want to find, the minimum periods p_{min} that MATR should have and the similarity *S*(*segment*, *motif*, *r*) where *segment* is the current ATR candidate, *motif* is the one next to the last motif in the segment and *r* is the similarity threshold, the detail of the MATR is as follows:

Algorithm MSATR

Input: DNA sequence *seq*, motif length scope (*a*, *b*), similarity threshold *r*, minimum periods p_{min};
output: MATRs

Begin
1 set k=a;
Repeat
2 Dividing
  1) Divide *seq* by length *k* in *k* times from index 0 to *k*-1, each time return an array of divided motifs;
  2) Return all the *k* arrays.
3 Joining
  for each array *M*, do
  1) let m_0 be the first motif in *M* and set *periods*=1, put m_0 in the *buffer*, and for each motif m_i in *M* ($0 < i < |M|$), do $f = S(buffer, m_i, r)$, and
  2) if f ==–1, *buffer*.add(m_i), *periods*++;
  3) else
    if (*periods* >= p_{min}), add *buffer* to *MATRs*;
    clear *buffer*, and *buffer*.add(m_i);
    periods=1;
  End for;
4 Clear *buffer*, *k*++;
Until *k* = *b*;
5 Output *MATRs*;
End

B. Complexity Analysis of the MSATR

According to the pseudo code above, the runtime of MSATR is related to the scope of the length of motif. The larger the scope is, the more time MSATR would cost. However, if the scope is limit (<<n), the time complexity of MSATR is linear. Since MSATR contain two steps (dividing and jointing), we can estimate the complexity separated.

If the set the length of motif is k, then MSATR needs dividing k times to find all MATRs with that motif length. And in i^{th} dividing, MSATR divides the sequences into $\lfloor (n-i)/k \rfloor$parts (n is the length of sequences), and put the motifs into arrays. In this process, the complexity is $k(\lfloor (n-i)/k \rfloor) < n$.

In jointing step, MSATR scans all motifs with k length in the arrays and compare each motif to the buffer. Hence, the complexity is still O (n). If the scope of the length of motif is m (m<<n), then MSATR's complexity is O (m(n+n)) = O (2mn) = O(n).

The space MSATR needs contains: the arrays to store n motifs from dividing step and the arrays to store MATRs which is detected from n motifs. So, the space complexity of MSATR is O (n).

C. Related Concept

According to the MSATR, most of the time cost lies in the repetitive similarity tests. And a large amount of the tests are vain, because large quantities of MATR candidates whose periods are less than p_{min} are not qualified.

Besides, some MATRs are not able to be found by the mechanism above. That is because when a certain motif *m* is not similar to the motif m_e in the MATR candidate, the

buffer is cleared and the finding starts over again with *m*. However, *m* could be similar to all the motifs after m_e and enough motifs after *m* itself, which makes a qualified MATR. For example, let p_{min} = 6, there is a MATR candidate $m_1m_2m_3m_4m_5$, and m_6 is not similar to m_2. However, m_6 is similar to $m_3m_4m_5$ and m_7m_8. In that case, $m_3m_4m_5m_6m_7m_8$ is a MATR which cannot be found by MSATR algorithm.

Considering the deficiencies above, this paper put forward the optimized algorithm *mMSATR* to reduce the runtime cost and improve the result quality.

To understand how algorithm *mMSATR* works, a new concept, which is essential to improve the runtime efficiency of algorithm, needs to be defined.

Definition 6 **CASM** (Chain of Adjacently Similar Motifs)[16]: For sequence $T = T_1T_2...T_p\ (p \geq 2)$, if the similarity of any two adjacent motifs T_i and T_{i+1} ($0 \leq i < p$) meets the similarity threshold *r*, then *T* is a CASM, and *p* is its length.

D. Algorithm mMSATR

In order to solve the deficiencies existing in MSATR algorithm, an optimal algorithm *mMSATR* adopts two tactics in the *dividing* and *joining* steps[16].

One of the major improvements is introducing the concept of CASM into the algorithm. It is obvious that being a CASM whose length is not less than the minimum periods p_{min} is a prerequisite for being a qualified MATR. By finding all the CASMs first, what is left is to find all the MATRs from the CASMs by almost the same mechanism of MSATR' *joining* step, which means the searching scope is cut down to the CASMs as shown in Fig. 3. In that case, less similarity tests need to be implemented in the second step, not only because the searching scope is reduced, but also because the adjacent motifs in CASMs are already proved to similar to each other. As a result, the runtime efficiency is improved.

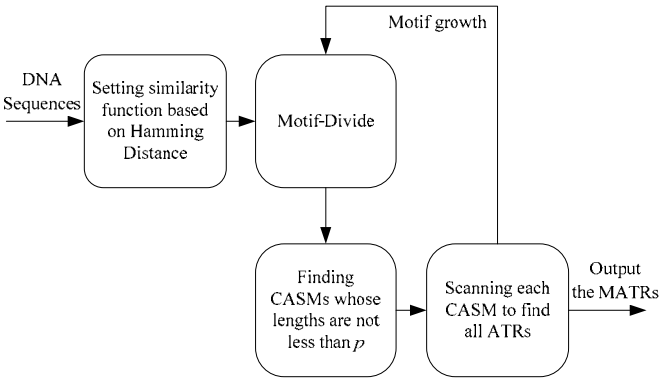

Figure 3. Workflow of the *m*MSATR Algorithm

Besides, when checking similarity between a new motif *m* and the motifs in the MATR candidate, MSATR compares *m* with every motif. However, when *m* is totally the same as one motif in the MATR candidate, the right thing to do is to stop the similarity test and return *true*, instead of continuing comparing *m* with the motifs left when it is obvious that the similarities between *m* and them will meet the threshold. This would benefit reducing runtime cost, when most motifs in the MATR are the same to one another.

Another major improvement is aimed to solve the problem of missing results. When algorithm *m*MSATR comes across motif m_c that is not similar with m_e in the MATR candidate, instead of clearing the buffer and starting over with m_c, *m*MSATR replaces buffer with a new MATR candidate $m_{e+1}\ldots m_{c-1}$ and checks m_c again. By doing such, no possible MATRs will be missed out. In order to know the index of m_e, the similarity function $S(buffer, m_c, r)$ needs to return the index *e* when m_c is not similar with m_e; and return -1 when no dissimilarity is detected.

By implementing the two major improvements in dividing and joining steps, *m*MSATR is able to reduce the searching scope and find more qualified MATRs. Algorithm *m*MSATR can be described as follows:

Algorithm *m*MSATR

Input: DNA sequence *seq*, motif length scope (*a*, *b*), similarity threshold *r*, minimum periods p_{min};
output: MATRs

Begin
1 set k=a;
Repeat
2 Dividing
for(*i*=0; *i*<*k*; *i*++) do
1) Starting from index *i*, divide *seq* by length *k*; let m_0 be the first motif divided from *seq*, set *length*=1, put m_0 in the *buffer*, and for each motif m_c divided from *seq* later(c starts from 1), do $f = S(m_{c-1}, m_c, r)$, and
2) if f ==–1, *buffer*.add(m_i), *length* ++;
3) else
if (*length* >= p_{min}), add *buffer* to *CASMs*;
clear *buffer*, and *buffer*.add(m_i);
length =1;
End for
4) Clear *buffer*, and return all the *CASMs*;
3 Joining
for each CASM *M* in *CASMs*, do
1) let m_0 be the first motif in *M* and set *periods*=1, put m_0 in the *buffer*, and for each motif m_i in *M* ($0 < i < |M|$), do $f = S(buffer, m_i, r)$, and
2) if f ==–1, *buffer*.add(m_i), *periods*++;
3) else
if (*periods* >= p_{min}), add *buffer* to *MATRs*;
replace *buffer* with motifs from m_{f+1} to m_{i-1};
periods= $i-f-1$, *i*--;
End for;
4 Clear *buffer*, *k*++;
Until *k* = *b*;
5 Output *MATRs*;
End

In the next section, experiments on SUA_SATR, MSATR and *m*MSATR would show that MSATR and

*m*MSATR is advanced in saving time and detecting more qualified MATRs.

IV. EXPERIMENTS AND RESULT ANALYSIS

To verify the improvements made by *m*MSATR works, experiments are set to evaluate the runtime efficiency and result number of different methods. We evaluate the quality for real-world data and synthetic data. The real-world data sets, for which the complete list of ATRs is not known, consisted of four genomes: chromosome I of yeast (230,203 bp) and the complete genomes of two types of E. coli: K–12 (4,639,221bp) and O157:H7 (5,498,450 bp) and the DNA sequence of Human's 22nd DNA chromosome from GenBank (http://www.ncbi.nlm.nih.gov/Genbank). The output quality is compared against SUA_SATR describe in [15]. The simulated data consisted of MATRs planted into a synthetic sequence of length 100,000 bp. For the simulated data we also compare the run times of the different programs.

As for the running environment, we use the Windows 7 OS with Intel(R) Core(TM) 2 Duo CPU P7350 @ 2.00GHZ and 2GB main memory, and the code was written on the platform of Eclipse 3.4.1 by Java.

A. *Evaluation of Searching Scope*

First experiment is conducted to verify that mMSATR is able to reduce the searching scope by introducing the concept of CASM, and how the scope changes with value p_{min}. In this experiment, we use the segment of the Human's 22nd DNA, and the size of the dataset is 200Kb, and the motif length k is set to 4, and similarity threshold r is 0.75.

In MSATR, the searching scope is all the motifs that divided by length k. While in *m*MSATR, as shown in Fig.4, when the p_{min} is small, like value 5, the scope has 261 CASMs, while the number of MATRs found is only 153. However, as the p_{min} increases, the number of CASM, the searching scope, keeps decreasing. When p_{min} reaches the value 10, the scope has 11 CASMs, out of which 10 MATRs are found.

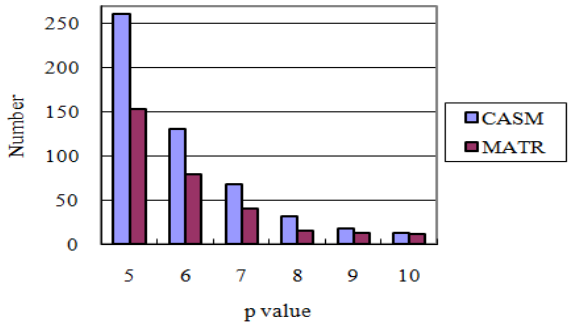

Figure 4. Evaluation of CASM and MATR.

Experiment has shown that the searching scope can be successfully reduced by finding all the CASMs in the *dividing* step of *m*MSATR; if the p_{min} increases, the scope will decrease corresponsively, and it is closer to the final results.

B. *Comparisons with Growing Dataset*

The next experiment is aimed to compare the runtime efficiency and results of SUA_SATR, MSATR and *m*MSATR algorithms on the growing dataset of Human's 22nd DNA chromosome.

In the experiment below, the minimum periods p_{min} is set to be 2, similarity threshold r is 0.75 and the motif length scope is (1, 80).

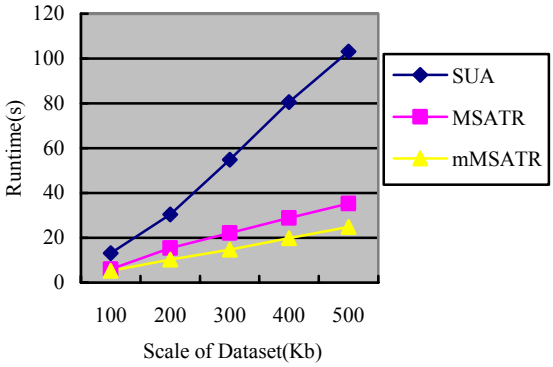

Figure 5. Evaluation of Runtime.

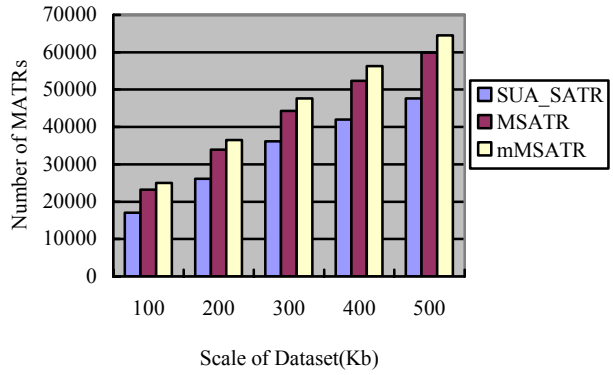

Figure 6. Evaluation of Result Number.

Firstly, we evaluate the runtime efficiency of MSATR, *m*MSATR and SUA_SATR with the increasing of dataset. Fig.5 shows that MSATR, mMSATR are much more efficient than SUA_SATR with the increasing of dataset. The larger the dataset is, the more time SUA_SATR spends. That is because SUA_SATR algorithm needs to compare each pair of motif even if they are not adjacent in the process of searching. So with the motif growth, SUA_SATR would spend more time to find MATRs. Compared to MSATR, *m*MSATR is more efficient. That is because the searching scope is reduced due to the introduction of CASM in *m*MSATR, less time is spent on the similarity test.

Secondly, according the improvement *m*MSATR made in the *joining* step, we compare the results of three algorithms. Fig.6 shows that MSATR and *m*MSATR found more MATRs than SUA_SATR in the growing datasets. That is because SUA_SATR can only search the MATRs with the special case in which the first element of each motif are the same. *m*MSATR is able to find more MATRs than MSATR in which those MATRs are missing.

To sum up, experiments have shown that MSATR and *m*MSATR are much more efficient than SUA_SATR with the increasing dataset in the same sequence. The larger the dataset is, the more obvious the superiority is. The main reason is that using the idea of Motif-Divide to avoid comparing motifs that are not adjacent in the process of searching MATRs. Besides that, MSATR and mMSATR can detect MATRs with different first letter motifs which cannot be found by SUA_SATR. Compared to MSATR, by using the concept of CASM, *m*MSATR reduces the searching scope and modifies the mechanism of MSATR to further improve the performance of MSATR.

C. Comparisons with Different Datasets

To test the quality of our algorithm's output in different cases, we ran on three real-world data sets: chromosome I of yeast (230,203 bp) and the complete genomes of two types of E. coli: K–12 (4,639,221bp) and O157:H7 (5,498,450 bp).

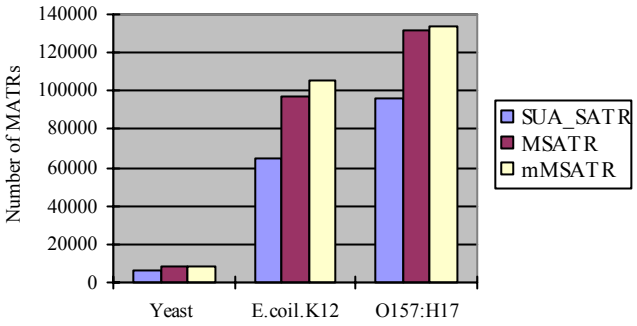

Figure 7. Number of MATRs found by SUA_SATR, MSATR and *m*MSATR on real data

In these experiments, the minimum periods p_{min} is set to be 2, similarity threshold r is 0.75 and the motif length scope is (1, 200). To avoid finding some highly short MATRs which may not carry any biological information, we limit the length of MATRs should longer than 10.

Fig.7 shows that MSATR and mMSATR found much more MATRs than SUA_SATR in the three genomic sequences, using MSATR's definition of MATR. We can see that even in different sequences, MSATR and *m*MSATR detect considerably more MATRs. Namely, there are plenty of MATRs with the different first letter motifs in different sequences which SUA_SATR cannot detect. Some examples are uniquely found by MSATR and mMSATR in these sequences: a motif of length 5 which repeats 5 times starting at position 241,403. In the E.coli–K12 genome, and 4 copies of a motif of length 17 in the E.coli–O157:H7 genome starting at position 358,919. In these cases, the MATRs did have different first letter motifs. Also, It can be seen that *m*MSATR is able to find more MATRs than MSATR.

D. Comparisions with Synthetic Dataset

We also use 10 synthetic sequences of length 100,000 to evaluate the algorithms. In the experiment below, the minimum periods p_{min} is set to be 2, similarity threshold r is 0.75 and the motif length scope is (1, 100).

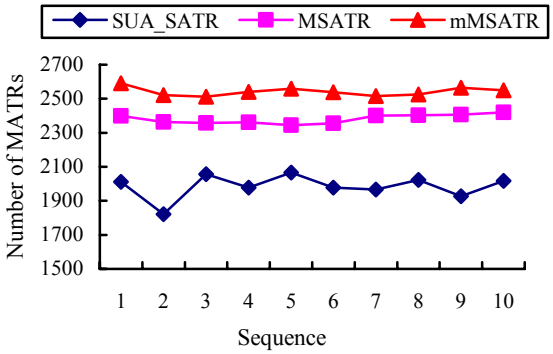

Figure 8. Number of synthetic MATRs found by SUA_SATR, MSATR and *m*MSATR on 10 sequences

When we ran three algorithms with the same parameters, the follow facts emerge as Fig. 8 shows: MSATR and mMSATR is able to find more MATRs while SUA_SATR often miss some with different first letter motifs; meanwhile, the runtime efficiency of algorithms MSATR and *m*MSATR outweighs that of the algorithm SUA_SATR.

We also note that these three algorithms have some redundancies in the result. There is a case like that: a MATR contains x motifs with length y, another MATR contains y motifs with length x, however these two MATRs may be the same.

E System Demonstration

Based on the work above, we developed a DNA Sequential Pattern Mining System (DSPMS) which implemented the function of mining tandem repeats. Fig. 9 is the main frame of the system.

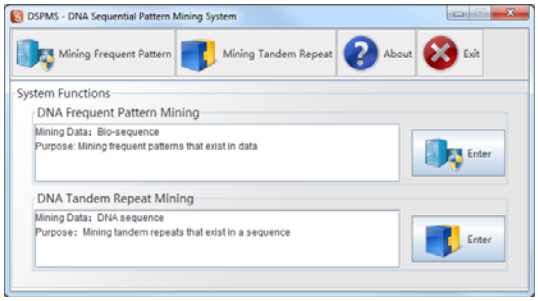

Figure 9. Mainframe of DSPMS

In the DSPMS system, the whole mining process is divided into several steps. The first step is to load data into the system as shown in Fig. 10, and then to configure the algorithm to implement the mining.

The system also can compare algorithms under the circumstance of different variables (Figure 11).

Then the results are presented to user by popping out a new dialog, containing results described by both text and graph.

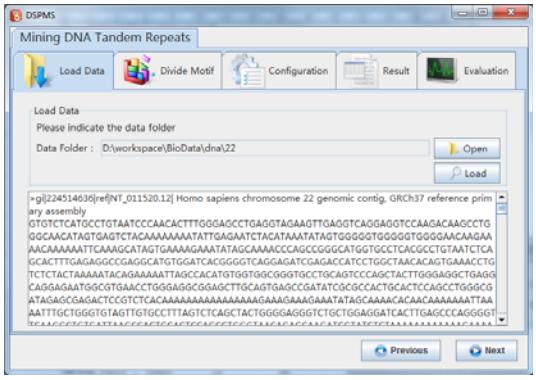

Figure 10. Load Data in the System

Figure 11. Results of comparison

V. CONCLUSIONS AND FUTURE WORK

In this paper, we describe a new model of which the essence is introducing the idea of Motif-Divide. Based on this model, a novel algorithm MSATR and an optimized algorithm *m*MSATR have been put forward for finding approximate tandem repeats. By introducing a new index structure based on Motif-Divide method, algorithm MSATR improves efficiency and result quality than SUA_SATR. Algorithm *m*MSATR introduced the concept of CASM to restrain the searching scope in order to promote its runtime efficiency. Besides, *m*MSATR adjusted the original mechanism in the *joining* step of MSATR to improve the deficiency of the result quality. Experiments have proved that this model is effectual, and the algorithms based on it have made positive effect as expected. Besides, we developed a DNA Sequential Pattern Mining System to facilitate future research. Future work is to find a more applicable similarity measurement for DNA sequence, so the ATRs we are trying to find can be more reasonable and useful in practical applications.

ACKNOWLEDGMENT

This work was supported by the National Natural Science Foundation of China under Grant No.10771176.

REFERENCES

[1] NM. Luscombe, D. Greenbaum, M. Gerste. What is bioinformatics? A proposed definition and overview of the field. Methods Information in Medicine, 2001, 40(4):346−358.

[2] Guo Shun, Guan Heshan, Jiang Qingshan. A novel algorithm for finding approximate tandem repeats in DNA sequences. Journal of Computer Research and Development. 2008, 45(Suppl.):175-179 (in Chinese).

[3] Zhu Yangyong, Xiong Yun. DNA Sequence Data Mining Technique. Journal of Software, 2007, 18(11): 2766-2781(in Chinese).

[4] Y. Li, A. Korol, T. Fahima, A. Beiles, and E. Nevo. Microsatellites: genomic distribution, putative functions and mutational mechanisms. Molecular Ecology, 2002, 11(12): 2453 -2465.

[5] Y. Kashi, D. King, and M. Soller. Simple sequence repeats as a source of quantitative genetic variation. Trends in Genetics, 1997, 13(2): 74–78.

[6] S.Beleza, C.Alves,A. Gonzalez-Neira. Extending STR markers in Y chromosome haplotypes. International Journal of Legal Medicine, 2003, 117(1):27-33.

[7] S.Gilmore, R. Peakall, J.Robertson, Short tandem repeat (STR) DNA markers are hypervariable and informative in Cannabis sativa: implications for forensic investigations. Forensic science international , 2003, 131(1): 65-74.

[8] Apostolico and F. Prefarata. Optimal off-line detection of repetitions in a string. Theoretical Computer Science, 1983, 22(3):297–315.

[9] Kolpakov R, Kucherov G. Finding maximal repetitions in a word in linear time. In: Proc. of the 1999 Symposium On Foundations of Computer Science. Washington: IEEE Computer Society, 1999. 596−604.

[10] M. Crochemore. An optimal algorithm for computing the repetitions in a word. Information Processing Letters, 1981, 12(5):244–250.

[11] M. Main and R. Lorentz. An O(nlogn) algorithm for finding all repetitions in a string. Journal of Algorithms, 1984, 5(3):422–432.

[12] Delgrange O, Rivals E. STAR: An algorithm to search for tandem approximate repeats. Bioinformatics, 2004, 20(16):2812−2820.

[13] G. Benson. Tandem repeats finder: a program to analyze DNA sequences. Nucleic Acid Research, 1999, 27(2):573–580.

[14] S. Kurtz, JV. Choudhuri, E. Ohleb usch, C.Schleiermacher, J. Stoye, R. Giegerich. REPuter: The manifold applications of repeat analysis on a genomic scale. Nucleic Acid Research, 2001, 29(22): 4633–4642.

[15] Wang D, Wang G, Wu QQ, Chen BC. Finding LPRs in DNA sequence based on a new index SUA [C]. In: Proc. of the IEEE 5th Symp. On Bioinformatics and Bioenginerering (BIBE 2005). Washington: IEEE Computer Science, 2005. 281−284.

[16] Li Sheng, Jiang Qingshan, Wei Dan. An optimized algorithm for finding approximate tandem repeats in DNA sequences. Second International Workshop on Education Technology and Computer Science. Wuhan, 2010. 68-71.

[17] Wang D, Zhao Y, Chen BC, Wang GR. SUA-Based algorithm for finding SATRs in DNA sequence. Journal of Northeastern University (Natural Science), 2007, 28(2):209−212 (in Chinese).

Qingshan Jiang is a professor at Xiamen University, China. He received a Ph.D. in Mathematics from Chiba University, Japan in 1996, and a Ph.D. in Computer Science from University of Sherbrooke, Canada in 2002. During his more than 25 years of study and research, he has published over 100 scientific papers in international journals and conference proceedings. His research interests include Pattern Recognition, Data Mining, Artificial Intelligence, and Bioinformatics

Sheng Li, born in China in 1984, received his Master Degree majored in Computer Software and Theory at Xiamen University in June 2010. His research includes Data Mining and is majorly focused on mining sequential patterns from bio-sequences.

Shun Guo, born in 1982, Ph.D. candidate at Xiamen University. Her main research interests include data mining and Bioinformatics

Dan Wei is a Ph.D. candidate at Xiamen University. Her current research interests include Artificial Intelligence, Data Mining and Bioinformatics.