Proceedings of the 28th IEEE
EMBS Annual International Conference
New York City, USA, Aug 30-Sept 3, 2006

Using Suffix Tree to Discover Complex Repetitive Patterns in DNA Sequences

Dan He*

Abstract— **The discovery of repetitive patterns is a fundamental problem in bioinformatics. It remains a challenging open problem because most of the existing methods, such as using annotated repeat database and extracting pairs of maximum repeated regions, can not give a correct definition incorporating both the length and frequency factors of the repetitive patterns. There is an algorithm considering both the pattern length and frequency. However, it could only find the simple "elementary" repeats and is not able to reveal the complex structure of the repetitive patterns. Furthermore, its time complexity $O(n^2 f)$, where n is the length of the sequence, f is the minimum frequency requirement, could be still too high for long DNA sequences. In this paper, we propose a novel algorithm using suffix tree to reveal the complex structure of the repetitive patterns in DNA sequences. We show that our algorithm achieves an $O(n^2/f^2)$ time complexity.**

Keywords: repetitive patterns, suffix tree, elementary repeats, complex structure.

I. INTRODUCTION

It has been shown that a remarkable fraction of the genome of complex organisms is repetitive patterns. Lewin [1] showed that more than 50% of the human genome consists of repetitive patterns. Although these repeats are normally regarded as "junk" regions, which do not contribute to the coding for proteins, they may be associated with various genetic diseases. Zheng and Lonardi [2] showed that quite a few human genetic diseases, such as fragile X syndrome, myotonic dystrophy and Friedreich's ataxia are related to the irregularities in the length of repeats. More cases can be found in [2]. Therefore the functions of these repetitive patterns are attracting more and more attentions and discovery of these repetitive patterns is of great importance.

The discovery of repetitive patterns is one of the fundamental problems in bioinformatics. There are two major methods to solve this problem. REPEATMASKER [3] uses an annotated library of repeats. This approach is not able to handle new genomes whose libraries of repeats are unavailable. The other method is to extract all pairs of similar repeated regions with maximal length [4], [5], [6]. They only look for pairs of repeats. However, in real biological sequences, the repetitive patterns often occur more than twice. It is more reasonable to incorporate frequencies of repeats.

Upon recognition of the importance of frequencies of repeats, several works [7], [8] were done to incorporate both the length and frequency factors into the definition of repetitive patterns. But their methods are complicated and difficult to analyze. Zheng and Lonardi [2] proposed a novel definition of repeats called "elementary repeats", which are the basic building blocks of longer repeats (called *complex repetitive patterns*). They also designed neat algorithms to find these elementary repeats. However, their algorithms are not able to retrieve the structures of the complex repetitive patterns. For example, their algorithms can not reveal which elementary block is contained in which complex repetitive pattern. Furthermore, it's time complexity $O(n^2 f)$, where n is the length of the sequence, f is the minimum frequency requirement, could be still prohibitive for DNA sequences as long as hundreds of thousands of bps, which is quite common for genome sequences.

In this paper, based on the definition of elementary repeats from Zheng and Lonardi [2], we propose an efficient algorithm to find the elementary repeats. By first finding all the possible candidates for the elementary repeats, then pruning those false positive candidates, we can achieve a more efficient algorithm, with time complexity $O(n^2/f^2)$, than the algorithm from Zheng and Lonardi [2]. We then extend our algorithm to reveal the structures of all complex repetitive patterns.

The outline of this paper is as follows: In Section II we summarize related work to the problem of finding repetitive patterns and we propose a novel algorithm for this problem in section III. We include our final remarks in Section IV.

II. RELATED WORK

Lots of work has been done on the problem of discovering repetitive patterns. However, it remains a challenging problem because most of the existing methods, such as annotated repeat database ([3]) and pairs of maximum repeated regions ([4], [5], [6]), can not give a correct definition incorporating both the length and frequency of the repetitive pattern.

In order to design a clean algorithm to discover repetitive patterns, a biological meaningful definition of repeats is necessary. Although this is not an easy task, some recent works have tried to take into account both the length and frequency factors of repeats ([7], [8]). However, the multiple sequence alignment strategy in the work by Bao and Eddy [7] and the A-Bruijin graph used by Pevzner et. al [8] are complicated and not easy to analyze.

Zheng and Lonardi [2] gave a novel definition of repeats, called *elementary repeats*, to incorporate both the length and frequency factors of repetitive patterns. Given the length threshold l and the frequency threshold f, they defined an elementary repeat of a sequence S as a substring A of maximal length greater than l, occurring at least f times (A is called $notrivial$). The frequency of every substring B

*D. He is with the Department of Computer Science, the University of Vermont, Burlington, VT 05405, USA hedanxkhc@hotmail.com

of A whose length is greater than l should be the same as the frequency of A. Every copy of B should occur with the same shift s in every copy of A (B is called a $subrepeat$ of A). Therefor the elementary repeats are actually the basic building blocks of the sequence. The containing of these elementary repeats in a long repeat results in complex structure of repetitive patterns. Zheng and Lonardi claimed that the discovery of the internal components of complex repeats can be helpful to infer the role of repeats. The conservation of these basic blocks could also reveal the evolutionary relationships among different types of repeats. Therefore we should first find these elementary repeats. If the elementary repeats are identical, we call them *exact elementary repeats*. If some differences among them are allowed, we call them *approximate elementary repeats*. Our algorithms are based on the definition of elementary repeats by Zheng and Lonardi [2].

The algorithm of Zheng and Lonardi uses a bottom-up strategy. They first use the linear-time algorithm by Gusfield [4] to build a suffix tree. Next they find the frequencies for all $n - l + 1$ seeds whose lengthes are l (These seeds are called *l-mers seeds*). These seeds start at positions from $S[1]$ to $S[n - l + 1]$, respectively. Then it merges the successive seeds with equal frequencies into $intervals$. After this step, it extracts all the intervals consists of seeds with equal frequencies greater than f (multiple copies of intervals will be included). Then for each interval, they check the corresponding occurrences of every pair of successive seeds, using the suffix tree. If every pair of successive seeds in this interval is also successive in all other copies, this interval is elementary repeat according to Lemma 5 in [2] (*A nontrivial substring A, which occurs at least twice, is an exact elementary repeat if and only if it is a maximal nontrivial substring such that all its l-mers are as frequent as A itself*). Otherwise this interval is divided into two pieces of new intervals to be checked again. Checking all the occurrences of a seed takes $O(n)$ time. The comparison of the relative positions for every pair of two successive seeds takes $O(f)$ time. Since each pair of successive seeds is checked at most once, and there are $O(n)$ pairs of such seeds, the time complexity of their algorithm is $O(n^2 f)$.

The above algorithm is not efficient. First, it needs to check all $n - l + 1$ seeds. However, most of them would be only one part of some elementary repeats, especially when the average length of elementary repeats is much greater than l, which is often the case. Second, it needs to check all copies of each interval. But we actually only need to check one copy of each interval. What's more, the algorithm can not reveal the structure for the complex repetitive patterns.

We first try to improve the algorithm for discovering exact elementary repeats. Our method extracts exactly one copy of possible candidates of elementary repeats. And then we try to prune the false positive candidates efficiently. We show our algorithm achieves a time complexity of $O(n^2/f^2)$ and in real applications it is even much faster. We next extend our algorithm to find the complex repetitive patterns. Our algorithm can report all the repetitive patterns which are nontrivial and what elementary repeats they contain, with the same time complexity $O(n^2/f^2)$. The biological meaning of these repetitive patterns will be left to the biologists.

Algorithm Suffix-ExactRepeat
Input: string S of length n, seed length threshold l, seed frequency threshold f
Output: the set $\mathbf{R}$ of all nontrivial exact elementary repeats in S

1. $\mathbf{T}, \mathbf{R} \leftarrow$ NULL (T is the candidate set)
2. traverse the suffix tree with breadth first search {
3. if the node C is nontrivial and left diverse {
4. $T \leftarrow T + \{c\}$ (c is the substring associated with node C)
5. $f(c) \leftarrow$ the frequency for substring c
6. block the search in the subtree of C
7. }
8. }
9. sort the $c_i \in T$ into $c_1, c_2, ..., c_{|T|}$ such that $l_1 \leq l_2 \leq ... \leq l_{|T|}$ (l_i is the length of substring c_i)
10. $R \leftarrow R + \{c_1\}$
11. for $i = 2$ to $|T|$ (candidate elementary repeats set)
12. build a suffix tree Tr_{c_i} for T_{c_i}
13. for $j = 1$ to $|R|$ (true elementary repeats set)
14. if for **all** j with f(R_{c_j})>f(T_{c_i}), **Suffix**(Tr_{c_i}, R_{c_j})=**false**
15. $R \leftarrow R + \{c_i\}$
16. if for some j, f(R_{c_j})>f(T_{c_i}), **Suffix**(Tr_{c_i}, R_{c_j})=**true**
17. report R_{c_j} is a no-subrepeat of T_{c_i}
18. remove R_{c_j} from T_{c_i}
19. check the remaining substrings of T_{c_i}
20. If they are of lengths greater than l and of frequencies greater than f
21. add them into T, at position i, in ascending order of their lengths
22. **Suffix**($Tree, S$) is a sub-procedure to check whether S is a substring in the suffix tree $Tree$
23. return R as the set of exact elementary repeats

Fig. 1. Algorithm Suffix-ExactRepeat to find the exact elementary repeats of a given string

III. Novel Algorithm for Discovering Repetitive Patterns

A. Discovery of Elementary Repeats

Due to the space limits, we only describe the algorithm of discovering exact elementary repeats. The idea can be easily extended to the cases of approximate elementary repeats. We first show our algorithm in pseudocode in Figure 1.

In line 2 to 8 of our algorithm, we find possible candidates of elementary repeats, then we filter those candidates which are not true elementary repeats. The extraction of candidates can be done in $O(n)$ time via suffix tree. Since an internal node in suffix tree must have at least two children, the frequency of a substring associated along the paths from the root to this node (we call *substring associated with this node*) is higher than the frequency of a substring associated with any child node of this node. Therefore, the substring

associated with the child node can not be an elementary repeat because at least it has a nontrivial prefix occurring more times than itself. We call a node in the suffix tree $nontrivial$ if there is a nontrivial substring associated with it. Therefore we can find the set of all possible candidates of elementary repeats, say T, by checking each internal node of the suffix tree at most once. If a node is nontrivial, we simply ignore its subtree since its frequency is higher than the frequencies of all its children. And also the left diverse nodes strategy [4] (A node v is called *left diverse* if at least two leaves in v's subtree have different left characters) excludes from the candidate set all the nontrivial suffixes of a candidate. Checking whether a node is left diverse can be done simultaneously as we check the frequency of the substring associated with this node.

A trivial way of pruning false positive elementary repeats is for each $i \in T$, we check i against all other candidates in T. If there is a candidate j being a substring of i but not a subrepeat of i, namely every copy of i contains a copy of j with the same shift, but j has more copies than i in S, then i can not be an elementary repeat. We call j a *no-subrepeat* of i. If there is no other candidate j being a no-subrepeat of i, we can safely make the conclusion that i is an elementary repeat. This trivial way would take $O(|T|^2)$ comparisons. We can improve it based on the observation that if j is a no-subrepeat of i then j must be a substring of i and the length of j must be less than i. Therefore we can sort these candidates in ascending order of their lengths. A candidate only needs to be proposed to those candidates with shorter lengths. Then we check these candidates in this ascending order and use a set R to store the currently found true elementary repeats. For each candidate we just need to check whether a candidate i has no-subrepeats in the set R. If yes, then it can not be a true elementary repeat and can be pruned, if no, then it is a new true elementary repeat and is added into R. In order to guarantee the correctness of this algorithm, when a candidate is judged as a false positive elementary repeat and it contains a no-subrepeat, we continue to check the remaining substrings of this candidate after removing the no-subrepeat from it, against the set R. Since their lengths are shorter than the current candidate, we just need to propose them to the current R. If they meet the criterions of elementary repeats, we add them into R. If they contain other no-subrepeats, we continue the process until either the length or the frequency criterion is not satisfied. We don't need to worry about the case that the remaining substrings may be no-subrepeats of some element in R. It can not happen since if it is the case this substring should have already been extracted as a candidate and have been included in the set T individually. The number of comparisons now is reduced to $O(|T|e)$, where e is the number of true elementary repeats and $e \leq |T|$. This is shown in line 11-22 in Figure 1.

Note that in line 9, we sort each $i \in T$ in ascending order of their lengths. This step is necessary to guarantee that each candidate will only be proposed to those currently found true elementary repeats. The first candidate after sorting must be an elementary repeat since there is no other candidate shorter than it. In line 10 we add the first candidate to the set of currently found elementary repeats. Then a new candidate would be compared with only those shorter candidates which have been checked to be true elementary repeats.

In line 12, for each candidate string, we build a suffix tree. When we compare a candidate with an elementary repeat, we first check their frequencies. If the frequency of the elementary repeat is less than that of the candidate, we know immediately that the elementary repeat can not be a no-subrepeat of the candidate and we can then propose this candidate to the next elementary repeat. Otherwise we check whether the elementary repeat meets the criterion of a no-subrepeat (being a substring of the candidate but has higher frequency than the candidate does). If yes, the elementary repeat is a no-subrepeat of the candidate. We can prune the candidate and check the next candidate. The time complexity to check one candidate against all true elementary repeats is $O(el')$, using suffix tree, where l' is the average length of the candidate repeats.

Therefore the time complexity of our algorithm is $O(ecl')$, where e is the number of elementary repeats, c is the number of candidates we found, l' is the average length of the candidate repeats, which is also the average length of the elementary repeats. However, we know that in the worst case, $e = n/(l'f)$ (we assume the elementary repeats are not overlapped with each other). And we could have at most n/f candidates since each of them occur at least f times and we only store one copy of them. Then we have the time complexity of our algorithm as $O(ecl') = O(\frac{n}{l'f}\frac{n}{f}l') = O(n^2/f^2)$ instead of $O(n^2 f)$ of the algorithm in [2].

We illustrate our algorithm by a simple example. Given the string $xabxaba$, we set $l = 1$ and $f = 2$. By our Suffix-ExactRepeat algorithm, the candidates are xab and a. Note that ab and b are nontrivial suffix of xab. They are pruned since they are not left diverse, namely the left symbols of every copy of ab are the same, x, and the left symbols of every copy of b are the same, a. On the contrary, a is a candidate since it is left diverse. a has two different left words x and b. xab is also left diverse since it has two different left words $\#$ and b (we add $\#$ at the beginning of the string and \$ at the end of the string). After we find all the candidates, we sort them according to their lengths. Then we first add a into the current found elementary repeat set. Then we propose xab to a. a is a substring of xab, however its frequency is higher than xab. Therefore a is a no-subrepeat of xab and xab is not an elementary repeat. Next we remove a from xab and we get x and b. Both of them meet the criterion of elementary repeat and they don't have any no-subrepeat. The elementary repeats are then x, a and b.

Gusfield [4] describes how to use suffix tree to find all maximal repeats in linear time. However, his definition of maximal repeats is different from ours here. According to him, if every copy of a substring is contained in another longer substring with the same shift, this substring can not be maximal repeat. Otherwise it is a maximal repeat. Therefore,

in the above example, ab and b can not be maximal repeats since they are all contained in xab with the same shift. a is a maximal repeat since it has one copy not contained by xab. xab is also a maximal repeat according to the definition by Gusfield. However, according to our definition of elementary repeat, it is not an elementary repeat since it has a no-subrepeat as a.

As we showed above, the time complexity of our Suffix-ExactRepeat algorithm is $O(n^2/f^2)$. However, in real applications, it is usually much more efficient. This is because in the real applications, e is usually much less than $n/(l'f)$, the number of candidates is also much less than n/f. Another important factor is the sorting strategy in line 9 and the frequency comparison strategy in line 14 of Figure 1 makes the pruning process much more efficient than our worst case scenario. For example, in the simulated experiments by Zheng and Lonardi [2], they have the parameters in average cases as $n = 20000$, $e = 10$, $f = 20$, $l' = 50$. Clearly the $O(f^3)$ factor will make a big difference between the time complexities of our algorithm and the algorithm of Zheng and Lonardi.

B. Discovery of Complex Repeats

Another advantage of our Suffix-ExactRepeat algorithm is that it is easy to be extended to a more general case: to reveal the structure of complex repetitive patterns. We show the detail of our Suffix-ExactComplexRepeat algorithm in pseudocode in Figure 2.

When we traverse the suffix tree, we should traverse through all the internal nodes to find all the nontrivial maximal repeats and store them in set T. However, we'd like to move those candidates who do not have nontrivial prefixes into set T'. This is done in line 10 to 12. We then apply the Suffix-ExactRepeat algorithm on candidates in T' to extract the set of true elementary repeats R. Next we propose the remaining candidates to R to identify all the no-subrepeats in them. In line 14-18, we need to propose each candidate in $T - T'$ to all the elementary repeats in R and report this candidate contains which elementary repeats as no-subrepeats. For example, for the string $xabxaba$, the algorithm Suffix-ExactComplexRepeat would first find the elementary repeats x, a, b and then report that xab contains x, a, b as no-subrepeats.

Surprisingly the time complexity of this algorithm is still $O(n^2/f^2)$ because the procedure is exactly the same as the Suffix-ExactRepeat algorithm. However, in real applications, obviously it takes much more time than the Suffix-ExactRepeat algorithm does since its candidate set T may be much larger and each candidate in $T-T'$ has to be compared with all elementary repeats.

Algorithm Suffix-ExactComplexRepeat
Input: string S of length n, seed length threshold l, seed frequency threshold f
Output: the set $\mathbf{R}$ of all nontrivial exact elementary repeats in S and reports of complex repetitive patterns

1. $\mathbf{T}, \mathbf{R} \leftarrow$ NULL
2. traverse the suffix tree with breadth first search {
3. if the node C is nontrivial and left diverse {
4. $T \leftarrow T + \{c\}$ (c is the substring associated with node C)
5. $freq(c) \leftarrow$ the frequency for substring c
6. $l_c \leftarrow$ the length of substring c
7. mark its subtree as non-elementary
8. }
9. }
10. for $c \in T$
11. if c is unmarked
12. $T' \leftarrow T' + \{c\}$
13. apply the **Suffix-ExactRepeat algorithm** (line 9 to 22) on T' and return true elementary set as R
14. for $i = 1$ to $|T - T'|$
15. build a suffix tree Tr_{c_i} for $(T - T')_{c_i}$
16. for $j = 1$ to $|R|$
17. report elementary repeat R_{c_j} as a no-subrepeat of $(T - T')_{c_i}$ if Suffix(Tr_{c_i}, R_{c_j})=**true** and f(R_{c_j})>f($(T - T')_{c_i}$)
18. Suffix($Tree, S$) is a sub-procedure to check whether S is a substring in the suffix tree $Tree$

Fig. 2. Algorithm Suffix-ExactComplexRepeat to find the exact elementary repeats of a given string

IV. CONCLUSION AND FUTURE WORK

In this paper, we proposed an efficient algorithm using suffix tree to solve one of the fundamental problems in bioinformatics: discovery of the complex repetitive patterns of DNA sequences. We try to incorporate the length and frequency factors into the process. Because of limit space, we only describe our algorithm for finding exact elementary repeats and the complex repetitive pattern containing these exact elementary repeats. However, similar ideas can be extended to the cases of approximate elementary repeats. We'd like to discuss algorithms for approximate elementary repeats in our future work.

REFERENCES

[1] B. Lewin, editor. *Genes VIII.* Oxford University Press, New York, NY, 2004.

[2] J. Zheng and S. Lonardi. Discovery of Repetitive Patterns in DNA with Accurate Boundaries. *Proc. of Fifth IEEE Symposium on Bioinformatics and Bioengineering (BIBE'05)*, pp. 105-112, Minneapolis, Minnesota, 2005.

[3] A. Smit and P.Green. http://www.repeatmasker.org

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

[5] S. Kurtz, J. V. Choudhuri, E. Ohlebusch, C. Schleiermacher, J. Stoye, and R. Giegerich. REPuter: The manifold applications of repeat analysis on a genomic scale. *Nucleic Acids Res. 29(22)*, PP. 4633-4642, 2001.

[6] S. Kurtz and C. Schleiermacher. REPuter: Fast computation of maximal repeats in complete genomes. *Bioinformatics, 15(5)*, PP. 426-427, 1999.

[7] Z. Bao and S.R.Eddy. Automated *De Novo* identification of repeat sequence families in sequences genomes. *Genome Research. 12(8)*, pp. 1269-1276, 2002.

[8] P.A.Pevzner, H.Tang, and G. Tesler. *De Novo* repeat classification and fragment assembly. *Proc. of Research in Computational Molecular Biology (RECOMB 2004)*, pp. 213-222, San Diego, CA, April 2004.