Discovery of Repetitive Patterns in DNA with Accurate Boundaries

Jie Zheng Stefano Lonardi
Department of Computer Science and Engineering
University of California, Riverside, CA 92521
{zjie, stelo}@cs.ucr.edu

Abstract

The accurate identification of repeats remains a challenging open problem in bioinformatics. Most existing methods of repeat identification either depend on annotated repeat databases or restrict repeats to pairs of similar sequences that are maximal in length. The fundamental flaw in most of the available methods is the lack of a definition that correctly balances the importance of the length and the frequency. In this paper, we propose a new definition of repeats that satisfies both criteria. We give a novel characterization of the building blocks of repeats, called elementary *repeats, which leads to a natural definition of repeat boundaries. We design efficient algorithms and test them on synthetic and real biological data. Experimental results show that our method is highly accurate.*

1. Introduction

Despite the incredible progresses that occurred in Genomics in the last fifty years, several fundamental questions remains unanswered. Some of the most intriguing questions are about the role of non-coding DNA, and in particular the role of repetitive sequences. As it turns out, a significant fraction of the genome of complex organisms is repetitive. For example, only about 1% of the human genome codes for proteins, but more than 50% of the human genome is composed of repetitive sequences [10]. Repeats are classified in five classes, namely pseudo-genes (0.1% of the human genome), simple repeats (3%), segmental duplications (5%), transposons (45%) and tandem repeats.

Although these long stretches of repeated DNA are commonly regarded as "junk", there is evidence that a variety of genetic diseases are associated with defects in the repeated structure of some regions of the human genome. More than a dozen human genetic diseases, including fragile X syndrome, myotonic dystrophy and Friedreich's ataxia are related with irregularities in the length of repeats. Insertional mutagenesis by SINEs (short interspersed elements) and LINEs (long interspersed elements) in mammals have resulted in cases of hemophilia, Duchenne muscular dystrophy, and sporadic breast and colon cancer.

A very important problem in computational biology is how to identify, classify, represent and visualize repeats. The general problem of repeat discovery and classification can be decomposed into two distinct subproblems. In the first, one identifies the boundaries of each copy of the repeats. This problem, usually attacked by performing local alignments, is difficult because *of degraded or partially deleted copies, related by distinct repeats, and segmental duplication covering more than one repeat* (quote from [1]). The second problem in the repeat classification is to infer the series of duplications that produced the final string. In this paper, we will focus on the first problem.

1.1. Previous work

Most of the computational tools for repeat identification available to biologists either require an annotated library of repeats (e.g., REPEATMASKER [15]) or simply output all pairs of repeated regions (e.g., REPUTER [9, 8], [14]). The problem with the first approach is that it is too limiting, in particular when trying to analyze a new genome for which a complete repeat library is unavailable. The latter approach is not completely satisfactory either, because it fails to elucidate the complex structure of repeats.

From a theoretical point of view, a repeat is usually defined as a pair of identical (or similar) substrings. For example, in Gusfield's definition [7] the objective is to find maximal repeated pairs. In REPUTER [9, 8] the output is again a list of pairs of similar strings of maximal length. Most definitions have in common the idea of maximizing the *length* of the repeats. They tend to ignore, however, the importance of repeat frequency, i.e.,

the number of occurrences.

As said above, repetitive elements typically occur more than twice in real biological sequences. For example, transposons typically occur hundreds of thousands of times in complex genomes. Therefore, we believe that a biologically meaningful definition of repeats must take into account both the length *and* the frequency of repeats (and perhaps other factors as well). Unfortunately, as noted by some researchers (e.g., [2]), devising a definition of repeats which is biologically plausible is not an easy task. When the frequency of repeats are allowed to be more than twice, there exist tools (e.g., [13], [3]) for finding short or tandem repeats. However, they are unable to find long and dispersed repeats.

Bao and Eddy [1] and Pevzner *et. al.* [11] recognized the shortcomings of definitions that rely solely on length, and attempted to take into account other factors. A rigorous definition of repeats considering both length and frequency, however, has not been given yet. The difficulty lies in the assessment of repeat boundaries once the objective of maximality in length is dropped. According to [1], approaches by multiple alignment are also problematic because the mosaic structure of repeats will be missed and the problem of multiple alignment itself is difficult. In [11], Pevzner *et. al.* proposed a graph called *A-Bruijn graph* to explore the mosaic structure of repeats. However, A-Bruijn graph is complicated and difficult to analyze, especially when the input sequences are long and contain a large number of repeats. In addition, the approach based on A-Bruijn graph still requires the pairwise local alignment of the input sequences given as input, and thus the results heavily depend on the performance of the pairwise alignment.

Recently, more attention has focused on the detection of repeat boundaries. Price *et. al.* [12] proposed the tool REPEATSCOUT to identify repeat boundaries via extension of consensus seeds. R. Edgar and E. Myers [4] developed the tool PILER to find repeats with reliable boundaries by considering well-known repetitive structures, e.g., terminal repeats and tandem arrays.

1.2. Our contribution

As noted by Bao and Eddy, *the problem of automated repeat sequence family classification is inherently messy and ill-defined and does not appear to be amenable to a clean algorithmic attack* (quote from [1]). We felt that in order to give a clean algorithmic attack, a natural and clean definition of repeats was indispensable. To this end, we propose a bottom-up approach that starts with the definition of basic building blocks of repeats that we call *elementary repeats*. Intuitively, an elementary repeat is a string whose substrings have similar (or identical) distribution of occurrences. Before giving the formal definition, let us consider an example in Figure 1 which illustrates the motivations as well as the basic idea.

In Figure 1, the DNA sequence contains three repeats S_1, S_2, and S_3, each occurring three times. At the first glance, it would appear that the segment $A = S_1S_2S_3$ is the "correct" repeat, as it is the longest. This is, in fact, what would be reported if the definition were solely based on maximal length. However, we can see that A is actually composed of independent "elementary" repeats, namely S_1, S_2, and S_3, because their order is shuffled in the third occurrence.

The example in Figure 1 is not a fictional scenario. Instead, it corresponds to a real biological case. In the four groups of LTR retrotransposons, Ty1/copia differs from Ty3/gypsy, Bel elements, and retroviruses by the order of POL protein coding domains. The order in Ty1/copia elements is *protease, integrase, RT/ribonuclease H*; while the order in the other three groups is *protease, RT/ribonuclease H, integrase*. The shuffling of the POL domains is one of the main features in the classification of LTR retrotransposons (see [6] for more details).

The identification of the internal components of complex repeats, such as POL domains of LTR retrotransposons, must be the first step in the accurate detection of repeats. The discovery of the internal structure could be also helpful to infer the functions of repeats because the order of these basic blocks is sometimes responsible for their functions. Moreover, conservation of the basic blocks among different types of repeats can reveal their evolutionary relationship.

The aim of this paper is to model the basic building blocks of repeats as combinatorial objects. To this end, we have defined a specific set of properties that they must satisfy. First, the blocks must occur a minimum number of times and can not be too short. Second, they must be *elementary*, i.e., they can not contain other basic blocks inside. We call these basic blocks *elementary* repeats. In this paper, we consider two types of elementary repeats, *exact* elementary repeats and *approximate* elementary repeats. The former type requires all copies of one repeat to be exactly identical, while the latter allows substitutions, insertions and deletions.

The rest of the paper is organized as follows. Section 2 introduces the notions of exact and approximate elementary repeats. In Section 3 we present efficient algorithms for *de novo* identification of both types of repeats. Then, we demonstrate the accuracy of identification with experimental results on both synthetic and real data. The last section summarizes the paper and outlines future work.

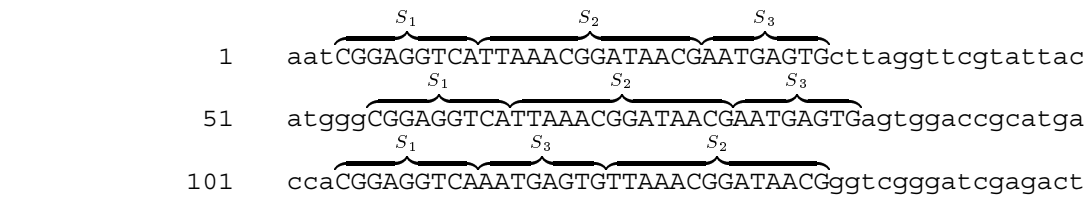

Figure 1. An example of elementary repeats.

2. Problem definition

We use standard concepts and notation about strings. The set Σ denotes a nonempty *alphabet* of *symbols*, and a *string* over Σ is an ordered sequence of symbols from the alphabet. We assume that the input is a string S of length n over Σ. We also use $|S|$ to denote the length of S. We write $S[i]$, $1 \leq i \leq n$, to indicate the i-th symbol in S. We use $S[i, j]$ as shorthand for the substring $S[i]S[i + 1] \dots S[j]$ where $1 \leq i \leq j \leq n$, with the convention that $S[i, i] = S[i]$.

Let A be a substring that occurs (exactly) multiple times in the input string S. Let $(A_1, A_2, \dots, A_m)$ be the sorted list of the occurrences of A. We call m the *frequency* of A. In the rest of the paper, let $f_m \geq 2$ denote the minimum frequency of repeats, which is a parameter set according to applications. Although the symbol A_i denotes the sequence composition of one of the occurrences of A, sometimes we will abuse this notation and used it to denote the position of A in S. The meaning will be clear from the context. We call A_i a *copy* of A in S.

Let B be a substring of A, and let $(B_1, B_2, \dots, B_k)$ be the sorted list of the copies of B in S. Clearly, every copy of A contains a copy of B as a substring, but B may also appear elsewhere in S. We say that B occurs with *shift* s in A if B starts at position $s + 1$ in A, that is $A[s + 1, s + |B|] = B$.

Definition 1 *Let A and B be substrings of S as defined above. Then B is a* subrepeat *of A if $k = m$ and every B_i occurs with the same shift s in A_i for all $i = 1, 2, \dots, m$.*

Intuitively, B is a subrepeat of A when the distribution of the occurrences of B in S "agrees" with that of A. Next, we need to consider the length of repeats. A substring of S is called *nontrivial* if its length is at least equal to a specified threshold l. Hereafter, we will always use l to denote such threshold. Typically, $l \approx \log_{|\Sigma|} n$, but l can be set as other values according to the application.

Definition 2 *An* exact elementary repeat *of S is a nontrivial substring A of maximal length such that A occurs at least f_m times and every nontrivial substring of A is a subrepeat of A.*

In other words, A is an exact elementary repeat when it does not contain any nontrivial substring with a different distribution of occurrences and it is maximal in length. Going back to the example in Figure 1, if $l = 8$, then S_1, S_2, and S_3 are exact elementary repeats because they do not contain any substring of length at least 8 with a different distribution of occurrences. However, the concatenation $A = S_1S_2S_3$ of the three substrings is not an exact elementary repeat, because nontrivial substrings S_1, S_2, and S_3 are not subrepeats of A.

Now let us generalize exact elementary repeats by considering the approximate case. Given a real number $\epsilon \geq 0$, we say that sequence A *ϵ-matches* sequence A' if $D(A, A')/|A| \leq \epsilon$, where $D(A, A')$ is the *edit distance* between A and A', i.e. the minimum number of edit operations (substitutions, insertions and deletions) that transform A into A'. We call A' as an *ϵ-copy* of A. In addition, let B be a substring of A. Then there must be a substring in A', denoted by B', which corresponds to B. We call B' the *image* of B. The concept of "image" mirrors the concept of "occurrence with shift s" in the exact case. Thus we will define it formally in the following. Given an edit script $E = (e_1, e_2, \dots, e_d)$ that transforms A into A', we say that E is *ordered* if, for any i, j such that e_i, e_j operate on $A[k]$, $A[l]$ respectively, $i < j$ implies $k < l$. That is, E operates on A from left to right.

Definition 3 *Suppose the shortest ordered edit script that transforms A into A' is $(e_1, e_2, \dots, e_d)$, where $(e_i, e_{i+1}, \dots, e_j)$, for $1 \leq i < j \leq d$, transforms a substring B of A into a substring B' of A'. Then, we call B' as the* image of B induced by A.

String B is called an *internal repeat* of A if (i) B is a substring of A, (ii) every image of B induced by A is an ϵ-copy of B, and (iii) the total number of non-overlapping ϵ-copies of B in S is equal to the images of B induced by A. Note that internal repeats should not be reported because they are not independent repeats.

Definition 4 *An* approximate elementary repeat *of S is a nontrivial string A of maximal length such that A has at least f_m ϵ-copies in S and every nontrivial substring of A is an internal repeat of A.*

In contrast to exact case, an approximate elementary repeat need not appear exactly in S. As a result, approximate repeats are much harder to identify than exact repeats. In the rest of the paper, we may sometimes omit the word *elementary* for brevity.

Before leaving this section, we emphasize that the number of copies of *all* exact repeats in S is upper bounded by n. In fact, for any two exact repeats A and B, A is not a substring of B; otherwise, A can be extended to B and remains an exact repeat, contradicting the maximality of length in definition 2. This is important because the size of output is a serious practical issue in repeat analysis.

3. Identification of elementary repeats

The goal in this section is to find both exact and approximate repeats in a long input sequence. A repeat can be represented by a list of pairs, each denoting the left and the right boundaries of a repeat copy. Hence, the problem of identifying repeats can be restated as the problem of finding their accurate boundaries. The task of our algorithms is to scan the input string S and decide whether a position of S is a repeat boundary or not.

3.1. Exact elementary repeats

An exhaustive algorithm for finding exact repeats is computationally impractical, because there are $\Theta(n^2)$ substrings in S, and each substring of length m contains $\Theta(m^2)$ substrings to check.

In sequence analysis the *q-gram* approach is well-known for its effectiveness in filtering out non-candidates (e.g., see [7] and reference therein). The idea is to collect the occurrences (or other statistics) for all the substrings of a given length q and use that information to discard substrings that can not be solutions to the problem at hand. Recently, the q-gram approach has been used in repeat analysis (e.g., [16], [4]).

Our definition of exact repeat allows the q-gram filtering approach for the identification of potential elementary repeats. More specifically, we collect statistics of l-mers in S, where l is the threshold that defines nontrivial substrings. We call these l-mers *seeds*.

We find the boundaries of exact repeats by detecting the positions in which there is a change in the distribution of the occurrences of the seeds. Given the definitions in Section 2, a necessary condition for a substring A to be an exact repeat is that all its nontrivial substrings are equally frequent. This condition allows us to reduce the search space of potential candidates considerably. Moreover, it suffices to consider only the frequencies of seeds, due to the following lemma.

Lemma 5 *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.*

Proof. If A is an exact repeat, then by definition all its nontrivial substrings, including those of length l, are as frequent as itself. If A is not an exact repeat, either it is not maximal in length, or it has a nontrivial substring B that is more frequent than A. But then any l-mer in B, which is also in A, is more frequent than A. ■

The above lemma allows us to compare only frequencies, rather than processing the occurrence lists, to rapidly discard most of the non-candidates. We just need to collect the statistics of l-mers in some efficient data structure, e.g., a hash table or a suffix tree, as shown in the pseudocode in Figure 2.

The algorithm EXACT-REPEAT works as follows. Lines 1-8 search for substrings of S whose seeds are equally frequent. We call these latter substrings *intervals* because they are represented by a pair of positions in S (corresponding to their first and last symbols). Lines 1-2 count the frequency of every seed using a suffix tree. The **for** loop of lines 4-6 identifies left and right boundaries by checking whether the frequencies of successive seeds change. Line 8 returns the intervals of seed frequency at least equal to f_m.

Finally, line 9 calls subroutine SEED-MERGE to check whether a selected interval is as frequent as its seeds. If so, by lemma 5 the interval is an exact repeat; otherwise, there must be some boundaries inside the interval that partition it into several exact repeats. To find these potential boundaries, we compare the occurrence lists of successive seeds. As shown in lines 12-15 in SEED-MERGE, if every pair of successive seeds have the same shift of one position in all other copies in S, then there is no additional boundary; otherwise, we insert the pair of seed positions as new boundaries.

Time and Space Complexity. To count the frequencies of all seeds in S, we build a suffix tree T_S for S in linear time (see e.g. [7] and references therein for details about suffix trees and their linear time construction), then for each seed w of length l, we count the number of leaves in the subtree corresponding to the string w, which is the frequency of the seed w. Therefore, we can find all intervals that consist of equally frequent seeds in time $O(n)$. Because the suffix tree requires linear space, the space complexity of lines 1-8 is also linear.

Algorithm EXACT-REPEAT(S, l, f_m)
Input: string S of length n, seed length l, minimum frequency f_m
Output: left and right boundaries of all exact elementary repeats in S that appears at least f_m times

```
1.   for i ← 1 to n − l + 1 do
2.       f_i ← frequency of the ith seed in S
3.   save 1, n as left and right boundaries, respectively
4.   for i ← 2 to n − l do
5.       if f_i ≠ f_{i−1} then save i as a left boundary
6.       if f_i ≠ f_{i+1} then save i + l − 1 as a right boundary
7.   pair consecutive left and right boundaries as intervals
8.   (I_1, I_2, ..., I_m) ← intervals whose seed frequency is ≥ f_m
9.   for i ← 1 to m do SEED-MERGE(I_i)
```

Subroutine SEED-MERGE ($\mathcal{I}$: an interval of seeds of equal frequency)

```
10.  for i ← 1 to |I| − l + 1 do
11.      P_i ← sorted list of occurrences of seed i in I
12.  for i ← 1 to |I| − l do
13.      for j ← 1 to |P_i| do
14.          if P_i[j] ≠ P_{i+1}[j] + 1 then
15.              insert i, i + 1 as new boundaries, go to line 4
```

Figure 2. A sketch of the algorithm that discovers exact elementary repeats

Line 9 takes time $O(n^2)$, as each pair of seeds is checked at most once and costs time $O(n)$. Here we assume that the number of occurrences of a seed is $O(n)$; however, the average number is much smaller. The space complexity of line 9 is $O(n)$, because we can obtain the occurrence list of each seed by looking it up in T_S when necessary rather than storing it explicitly.

3.2. Approximate elementary repeats

Compared with exact case, approximate repeats are more realistic for applications in molecular biology. However, the problem of finding approximate repeats precisely is very difficult, therefore we give a heuristic which extends the ideas for finding exact repeats to follow the definition for approximate repeats.

The basic idea for finding approximate repeats is the following. Suppose a repeat A ϵ-matches $A_1, A_2, \ldots, A_m$ in S, and two seeds of A, say H and J, occur in A with shifts s_H, s_J respectively. Then the images of H induced by A are likely to occur in $A_1, \ldots, A_m$ with shifts close to s_H. Similarly, images of J are likely to occur in the copies of A with shifts close to s_J. Thus, in each copy of A the offset of the images of H and J is close to $s_H - s_J$. If we subtract the offset, then the occurrence lists of H and J shall be similar, where the similarity is measured by number of images of H that are close to images of J and vice versa. In our algorithm, we use the similarity between the occurrence lists of two *successive* seeds to measure the likelihood that they belong to the same approximate repeat.

For example, S = `cACGTGagACGAGgcaACGTG`, where the capital letters represent repeat `ACGTG` which occurs three times. Suppose the length of seed is two. The occurrences of seed `AC` are 2, 9, and 17, and the occurrences of seed `CG` are 3, 10, and 18. The two seeds have similar occurrence lists except for a shift of one position, hence they are likely to belong to the same repeat. Notice the substitution in the second occurrence `ACGAG`, which is acceptable in our heuristic.

The algorithm APPROXIMATE-REPEAT, which is sketched in Figure 3, works as follows. In line 1, we locate the blocks consisting of seeds that are at least as frequent as a specified threshold f_m. This step reduces the search space by discarding non-repetitive regions. The **for** loop of lines 2-4 merges successive seeds in each selected block into longer substrings if their occurrences are similar. We call these longer substrings, which are constructed by merging one or more successive seeds, *contigs*. Similarly, line 5 merges successive contigs if they belong to the same repeat.

In the subroutine APX-SEED-MERGE, we decide whether a pair of successive seeds should be merged based on the similarity scores of their occurrence lists. Note that the scores and the thresholds are related with the order of seeds. The subroutine SIM-SCORE calculates the similarity score, by first removing the offset in line 11. Then in the **for** loop of lines 13-16, we count the number of positions in P that have a close position in Q. The *pal* of P_i is the position Q_j in Q with the smallest absolute difference with P_i. If the difference is no more than a specified threshold d_m, then we increase

bonus b by one; otherwise, we increase penalty p by one. The similarity score is defined as $b - \ln p$, a heuristic formula that emphasizes the significance of matches. The heuristic threshold $SS_{\min}(P, Q)$ is defined as a fraction of $|P|$.

The subroutine CONTIG-MERGE is similar to APX-SEED-MERGE, except that now we are trying to merge the last seed of the previous contig with the first seed of the next contig. The purpose of CONTIG-MERGE is to reconnect segments that are broken by substitutions and indels.

Time and Space Complexity. It takes $O(n)$ time to find the blocks that consist of seeds of frequencies at least f_m. For each distinct seed, SIM-SCORE is called $O(n)$ times, as in APX-SEED-MERGE and CONTIG-MERGE. Each call of SIM-SCORE takes time linear in the size of input occurrence lists. Therefore, the total running time of algorithm APPROXIMATE-REPEAT is $O(n^2)$. On average, however, it is much faster, as most seeds are likely to occur only a few times. The algorithm takes again linear space.

4. Experimental results

In this section, we demonstrate the accuracy of our algorithms by showing some experimental results. We test our algorithms on synthetic datasets obtained by inserting simulated and real biological repeats into random DNA sequences. We do not compare our results with other tools for repeat identification, because the concept of elementary repeats is unique to this approach while other tools (e.g. REPUTER) output pairs of maximal repeats which are incomparable with our output.

4.1. Finding simulated repeats

Our method of constructing the simulated DNA sequence S is as follows. First, we generate a set of random DNA strings, which corresponds to a set of elementary repeats. Each repeat has attached a specified multiplicity which is generated by Poisson distribution. We randomly permute the multiset of repeat copies by the algorithm of Fisher and Yates [5]. Then, we alternate the repeat copies (selected according to their order in the permutation) with purely random DNA sequences whose lengths are generated by Poisson distribution. The frequencies of repeats and the lengths of the gaps among repeats are selected according to Poisson distribution because it is the most appropriate probabilistic model in this case (see, e.g. [17]).

When we are simulating approximate repeats, we also randomly mutate each copy before it is used. The sequence of edit operations is generated randomly, and the number of mutations is bounded by a percentage, say 2%, of the length of the repeat copy. Finally, we append at both ends of the assembled sequence two pieces of random DNA, and that completes the construction of S. We also record the left and the right boundaries of each repeat copy, which will be compared with the output of our algorithms.

When we feed S as input to our algorithms, the output is a list of pairs of boundaries. A boundary in the output matches a boundary in the input if they are within 5 bases of each other. An output repeat copy matches an input one if *both* left and right boundaries match that of the input copy. The *sensitivity*, denoted by S_e, is defined as the ratio of the number of output copies matching the input over the total number of input repeat copies. The *true positive rate*, denoted by T_p, is defined as the ratio of the number of output copies matching the input over the total number of output copies.

In the tables in Figure 4, we show the accuracy of our algorithm on exact repeats of average lengths 50 and 200 respectively. Similarly, tables in Figure 5 report the results for approximate repeats. In each table, we carried out 5 runs with gap lengths ranging from 50 to 250 as shown in the tables; for each run, we executed our program 100 times. In every execution, the average number of distinct repeats is 10, the average frequency is 20, and the seed length is 15. Each row of the tables summarizes the results of one run, namely the worst, the average, and the best accuracy of the 100 tests of the run.

From the tables, we can make the following observations. First, the accuracy is remarkably high. The average value of S_e is higher than 98% and the average value of T_p is higher than 95%, except for the approximate repeats of average length 200. Longer approximate repeats are harder to detect, as the edit operations may occur in a narrow interval. In practice, however, elementary repeats are unlikely to be that long. The reason that the accuracy of exact repeat detection is not 100% is that the random DNA strings we generated are not "random" enough. As a result, there are additional repetitive patterns and fragmented repeats, which are not counted in validation but have been captured by our algorithm. Similar situation occurs in approximate case.

Second, the performance of the algorithm for approximate repeats is more consistent than that for exact repeats. This is due to the fact that the algorithm for approximate repeats is more adaptive and thus able to handle more noisy input.

Third, in many cases, especially for approximate repeats, the value of S_e is close to T_p. This happens because often the algorithm detects one correct boundary while the other boundary falls outside 5 bases of the

Algorithm APPROXIMATE-REPEAT(S, l, f_m)
Input: string S of length n, seed length l, minimum repeat frequency f_m
Output: left and right boundaries of approximate repeats in S
1. $(B_1, B_2, \ldots, B_k) \leftarrow$ blocks of seeds of frequencies $\geq f_m$
2. **for** each selected block B_i **do**
3. APX-SEED-MERGE (B_i)
4. save contigs from APX-SEED-MERGE into set C
5. CONTIG-MERGE (C)

Subroutine APX-SEED-MERGE ($\mathcal{B}$: a block of seeds of frequencies $\geq f_m$)
6. **for** $i \leftarrow 1$ **to** $|\mathcal{B}| - 1$ **do**
7. $(P, Q) \leftarrow$ sorted lists of occurrences of seed $(i, i+1)$
8. **if** SIM-SCORE $(P, Q, 1) \geq SS_{\min}(P, Q)$ **or**
9. SIM-SCORE $(Q, P, -1) \geq SS_{\min}(Q, P)$ **then**
10. merge seeds i, $i+1$

Subroutine SIM-SCORE (occurrence lists P and Q, offset d)
11. $Q \leftarrow Q - d$
12. $b, p \leftarrow 0$ /* bonus b and penalty p */
13. **for** $i \leftarrow 1$ **to** $|P|$ **do**
14. $Q_j \leftarrow$ *pal* of P_i in Q /* Q_j is the closest to P_i in Q*/
15. **if** $|P_i - Q_j| \leq d_m$ **then** $b \leftarrow b + 1$
16. **else** $p \leftarrow p + 1$
17. **return** $b - \ln(p)$

Subroutine CONTIG-MERGE (list of contigs C)
18. **for** $i \leftarrow 1$ **to** $|C| - 1$ **do**
19. $A \leftarrow$ last seed of contig C_i
20. $B \leftarrow$ first seed of contig C_{i+1}
21. $d_{AB} \leftarrow$ position offset of B against A
22. **if** $d_{AB} \leq D_{\min}$ **then** /* A, B are close enough */
23. $(P, Q) \leftarrow$ sorted occurrence lists of seeds (A, B)
24. **if** SIM-SCORE $(P, Q, d_{AB}) \geq SS_{\min}(P, Q)$ **or**
25. SIM-SCORE $(Q, P, -d_{AB}) \geq SS_{\min}(Q, P)$ **then**
26. merge contigs C_i, C_{i+1}

Figure 3. A sketch of the algorithm that discovers approximate elementary repeats

correct one, and it causes a false positive and a false negative simultaneously. These false positives, however, have long overlaps with correct repeats.

4.2. Finding real biological repeats

We also test our algorithms on synthetic data when the repeats are true biological repeats, i.e., we insert copies of real repeats into synthetic DNA sequences. We choose the well-known *Alu* repeats, a family of short interspersed elements (SINEs) that comprises roughly 10% of the human genome. When exact copies of Alu are inserted into random DNA sequences, the average accuracy of our method is above 96%. In approximate case, the average accuracy is above 80%. Due to space limitation, we leave the comprehensive analysis of real repeats to the journal version of this paper.

5. Conclusion and future work

In this paper, we have proposed a novel characterization of repetitive patterns in DNA sequences, which leads to a natural definition of repeat boundaries. It is a bottom-up strategy, starting with the basic building blocks of repeats called elementary repeats. We give rigorous definitions of exact and approximate elementary repeats, and design two efficient algorithms for *de novo* identification of both types of repeats with high accuracy. To our best knowledge, it is the first time that both length and frequency are considered in finding repeats.

A promising future direction would be the identification of complex repeated structures which are composed of elementary repeats as building blocks. It is our hope that complex repeats would correspond to biologically meaningful repeats in the genomic sequences.

input	output					
gap	S_e (%)			T_p (%)		
ave.	worst	ave.	best	worst	ave.	best
50	92.8	99.9	100	86.3	99.9	100
100	94.7	99.9	100	90	99.8	100
150	100	100	100	100	100	100
200	100	100	100	100	100	100
250	90.4	99.9	100	82.5	99.8	100

input	output					
gap	S_e (%)			T_p (%)		
ave.	worst	ave.	best	worst	ave.	best
50	89.0	99.8	100	77.4	99.7	100
100	94.3	99.9	100	89.3	99.7	100
150	100	100	100	100	100	100
200	87.4	99.2	100	69.6	98.5	100
250	100	100	100	100	100	100

Figure 4. Accuracy of exact elementary repeats detection in simulated data when the average length of repeats is 50 (table on the left) and 200 (table on the right)

input	output					
gap	S_e (%)			T_p (%)		
ave.	worst	ave.	best	worst	ave.	best
50	95.8	98.8	100	94.6	98.7	100
100	96.6	99.3	100	96.6	99.3	100
150	96.8	99.1	100	96.8	99.0	100
200	94.3	98.0	100	94.3	97.9	100
250	95.1	98.6	100	95.1	98.5	100

input	output					
gap	S_e (%)			T_p (%)		
ave.	worst	ave.	best	worst	ave.	best
50	80.2	87.2	94.1	78.9	85.5	94.1
100	81.6	87.5	93.1	78.8	86.1	93.1
150	83.2	87.5	91.9	80.7	85.8	91.5
200	79.7	87.4	92.7	75.3	85.8	92.7
250	80.9	87.5	95.4	78.0	85.8	94.3

Figure 5. Accuracy of approximate elementary repeats detection in simulated data when the average length of repeats is 50 (table on the left) and 200 (table on the right)

References

[1] Z. Bao and S. R. Eddy. Automated *De Novo* identification of repeat sequence families in sequenced genomes. *Genome Research*, 12(8):1269–1276, 2002.

[2] G. Benson. An algorithm for finding tandem repeats of unspecified pattern size. In S. Istrail, P. Pevzner, and M. Waterman, editors, *Proceedings of the 2nd Annual International Conference on Computational Molecular Biology*, pages 20–29, New York, NY, 1998. ACM Press.

[3] G. Benson. Tandem repeats finder – a program to analyze dna sequences. *Nucleic Acids Res.*, 27:573–580, 1999.

[4] R. Edgar and E. Myers. Piler: identification and classification of genomic repeats. In *Proc. of the 13th International Conference on Intelligent Systems for Molecular Biology (ISMB'05)*, page To appear, Detroit, Michigan, 2005. AAAI press, Menlo Park, CA.

[5] R. A. Fisher and F. Yates. *Example 12, Statistical tables*. London, 1938.

[6] E. Galun. *Transposable elements: a guide to the perplexed and the novice with appendices on RNAi, chromatin remodeling and gene tagging*. Kluwer academic, 2003.

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

[8] 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):4633–4642, 2001.

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

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

[11] P. A. Pevzner, H. Tang, and G. Tesler. *De novo* repeat classification and fragment assembly. In *Proc. of Research in Computational Molecular Biology (RECOMB)*, pages 213–222, San Diego, Ca, April 2004.

[12] A. L. Price, N. C. Jones, and P. A. Pevzner. De novo identification of repeat families in large genomes. In *Proc. of the 13th International Conference on Intelligent Systems for Molecular Biology (ISMB'05)*, page To appear, Detroit, Michigan, 2005. AAAI press, Menlo Park, CA.

[13] M. Sagot and E. W. Myers. Identifying satellites in nucleic acid sequences. In S. Istrail, P. Pevzner, and M. Waterman, editors, *Proceedings of the 2nd Annual International Conference on Computational Molecular Biology*, pages 234–242, New York, NY, 1998. ACM Press.

[14] J. P. Schmidt. All highest scoring paths in weighted grid graphs and their application to finding all approximate repeats in strings. In *Proceedings of the 3rd Israel Symposium on Theory of Computing and Systems*, pages 67–77. IEEE Computer Society Press, 1995.

[15] A. Smit and P. Green. REPEATMASKER. Available at `http://www.repeatmasker.org/`.

[16] P. E. Warburton, J. Giordano, F. Cheung, Y. Gelfand, and G. Benson. Inverted repeat structure of the human genome: the x-chromosome contains a preponderance of large, highly homologoous inverted repeats that contain testes genes. *Genome Research*, 14:1861–1869, 2004.

[17] M. S. Waterman. *Introduction to Computational Biology*. Chapman & Hall, 1995.