A Novel Algorithm for Finding Interspersed Repeat Regions

Dongdong Li*, Zhengzhi Wang, and Qingshan Ni

College of Mechatronics Engineering and Automation, National University of Defense Technology, Changsha 410073, China.

The analysis of repeats in the DNA sequences is an important subject in bioinformatics. In this paper, we propose a novel projection-assemble algorithm to find unknown interspersed repeats in DNA sequences. The algorithm employs random projection algorithm to obtain a candidate fragment set, and exhaustive search algorithm to search each pair of fragments from the candidate fragment set to find potential linkage, and then assemble them together. The complexity of our projection-assemble algorithm is nearly linear to the length of the genome sequence, and its memory usage is limited by the hardware. We tested our algorithm with both simulated data and real biology data, and the results show that our projection-assemble algorithm is efficient. By means of this algorithm, we found an un-labeled repeat region that occurs five times in *Escherichia coli* genome, with its length more than 5,000 bp, and a mismatch probability less than 4%.

Key words: repeats, random projection, assemble

Introduction

It is known that more than 50% of the human genome sequences are reiterated ones (*1*). This phenomenon also appears in many other species. A lot of secrets are hidden in these repetitive fragments, and many researchers are studying on this topic. The first step of this research is to find all repeats under a given standard in the complete genome.

Repeats in DNA are commonly classified into tandem and interspersed repeats (*2*). The tandem repeats are the characters of eukaryotes, which seldom exist in prokaryotes (*2*). These repeat units connect together and reiterate several thousand times to form a concentrated region. The units of interspersed repeats usually distribute in the whole genome. They are further classified into four DNA elements: long interspersed nuclear elements (LINE), short interspersed nuclear elements (SINE), long terminal repeats (LTR), and DNA transposon. There are many distinctions between tandem and interspersed repeats, and the identifying methods are different. A few works have been published to accomplish this task. For the problem of tandem repeats finding, Beason (*3*) realized a program called Tandem Repeats Finder, whose fundamental idea is similar to BLAST. It first collects the information of the short exact repeated fragments (seeds), and then extends them to approximated tandem repeated units based on some statistical criteria. There are some other methods that adopt the similar seed-extending idea (*4*), but their criteria are of pattern structures. Kolpakov *et al* (*5*) realized a very efficient combinatorial algorithm named mreps. It can find tandem repeats in a single run and has no limitation on the size of patterns. However, on the problem of interspersed repeats finding, there is very small amount of efficient methods proposed. A simple algorithm is to find the diagonal in a dotted graph (*6*). Some heuristic methods can also find interspersed repeats, however, the deficiencies of these methods are the huge computer resource usage and the high miss rate. Kurtz *et al* (*7*) developed an efficient program called REPuter that could find several types of repeats within the interval, which is a polynomial function of time. However, this method is based on the comparability of two sub-sequences, while it is hard to find the repeats that occur many times in the DNA sequences, which is unfortunately the style that long interspersed fragments usually appear.

The repeats finding problem can be treated as the motif finding problem. Many researchers have been working on the motif finding problem, and some algorithms have been proposed. The most commonly used algorithms are TEIRESIAS (*8*), CONSENSUS

*** Corresponding author.**
E-mail: li_dong_dong@yahoo.com.cn

This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

(*9*), WINNOWER (*10*), random projection (*11*), and so on. Their target is to find short conserved fragments in DNA sequences. For example, the typical problem is to search motifs of length 15 to 20 bp from sequences of length 600 bp, with the maximum of four mismatches permitted. However, in typical repeats finding problems we have to find the fragments with length of several hundred to several thousand base pairs from DNA sequences with length of several million to several hundred million base pairs. If these methods are employed directly, the computing is unpractical. Besides, there are many in-del errors in the repeats, while the motif finding algorithms are sensitive to in-del errors. These problems also restrict the application of these algorithms.

Repeats can be considered as the combination of several short conserved fragments. If we can collect the information of these segments, we should be able to find the repeats by assembly. Based on this idea, we propose the projection-assemble algorithm to find the long interspersed repeats in DNA sequences. We tested our algorithm with both simulated and real genome data. The results show that our algorithm is efficient and effective.

Algorithm

We define the problem of long interspersed repeats finding as: given a DNA sequence S of length L, find the repeat R with its length larger than l; the error probability should be less than p and occurrence is larger than m. Our algorithm consists of two steps. In the first step, we collect all fragments with length n (less than l), which meet some statistical conditions. In the second step, we assemble the fragments obtained in the first step and export the repeats.

Fragment Collection

In this stage, we try to get a fragment collection that includes as many fragments as possible. The fragments in this collection will be part of the repeat R. We also set some criteria to remove the irrespective segments. In fact, this is a problem of fixed-length motif finding. Due to the large length of S, these motifs are very subtle and are difficult to find.

Usually, the occurrence of fragment A in sequence S is a binomial distribution, and the expectation value is (for the convenience, we suppose that the four bases have the same occurrence probability, noted as q, and at each position these bases have independent occurrence probability):

$$E\left(N_{AS}\right) \approx (L-n+1)\, q^{n} \tag{1}$$

Here, N_{AS} is the number of occurrence of fragment A in sequence S, L is the length of S, and n is the length of A. There is small error in formula (1), however, in most cases L is quite large and $n \ll L$, so this formula is still a good approximation.

If A occurs once in repeat R and, as we have supposed, repeat R occurs m times in sequence S, we can write the expectation as:

$$E\left(N_{AR}\right) \approx (L-n+1)\, q^{n} + m\,(1-p)^{n} \tag{2}$$

Here, N_{AR} is the number of occurrence of fragment A, A appears in repeat R once and R occurs m times in sequence S, p is the error probability of repeat R. Under this condition, the distribution of the length of fragment A is still a binomial distribution, and it has the same shape as the previous one.

The only difference in formulas (1) and (2) is the item $m\,(1-p)^{n}$. We use $N_0 = \frac{(N_{AS}+N_{AR})}{2}$ as the threshold to distinguish these two distributions. From formula (2) we can see that the larger the n, the less difference between the two distributions. Because for larger n, it needs more computation time and occupies more memory, so it is necessary to restrict the range of the fragments' length. On the other hand, if n is too small, many random fragments could be added into the collection so that it increases the computation dramatically. This is a serious problem when we plan to assemble these fragments into a long repeat, while small n can't guarantee the correctness of the assembly. We did a simple computation on our computer, which showed that when n was larger than 9, the memory usage exceeded our computer capacity. But it is impossible to complete the assembly with fragments of length less than 9. Therefore, we must find a method to solve this problem.

Based on Buhler's random projection algorithm (*11*), we propose a metabolic method for the repeats finding problem. We collect fragments of length n_0, which is large enough to perform assembly. When we do statistics, we do not select these fragments directly, instead, we randomly select n_1 positions from these fragments and perform statistic calculation for these projected fragments. For other positions, we use a position weight matrix to save the bases' occurrence time. Figure 1 is a sketch map of such operation.

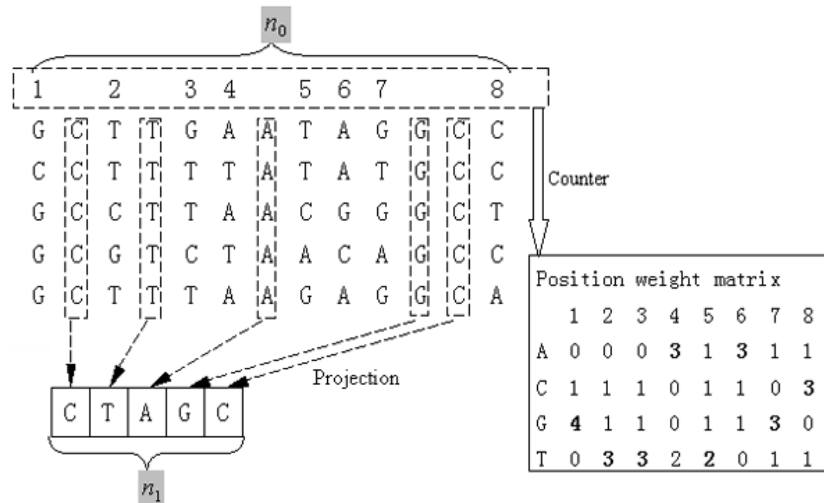

Fig. 1 Projection and position weight matrix.

After the projection and statistic, we can divide all fragments of length n_0 in sequence S into 4^{n_1} classes; each class has an assistant position weight matrix. So, if most fragments in one class are instances of a common fragment (with some errors), the assistant matrices should also have a consensus sequence. On the other hand, if most fragments in one class do not belong to a common fragment, the assistant matrices will be the same as the background distribution. Therefore, we can use these assistant matrices as the criterion to select the fragments that are more likely to be part of a repeat. In our algorithm, we use the following information entropy as criterion:

$$Entropy = \sum En\,(i) \tag{3}$$

Here, $En\,(i)$ means the information entropy in position i, and the sum is for all positions that have not been projected (in fact, this sum can also be calculated from all bases in this matrix, because the projected positions have the same bases and their entropy is 0). Given a threshold $Entropy^0$, we select the classes with large entropy (as a precondition, each of these classes must include enough items). They are the candidates for next step.

The following is the detail of the fragment selection algorithm:

Begin.

Set candidate collection M to be null.

Reiterate the following steps F times (do F times random projection)

Set fragment set G to be null.

Generate n_1 $(1 \le n_1 \le n_0)$ un-repeated numbers randomly, noting as g.

Inner recycle. For all $L - n_0 + 1$ fragments of S, do the following operation:

Construct fragments of length n_1 (called as B) to satisfy the equation $B\,[i] = A\,[g\,(i)]$.

If $B \in G$

Modify the statistic parameters and the position weight matrix.

Else

Add a class into collection G and set the related parameters.

Inner cycle stops here.

Fragment filtration. For each class in collection G, if it includes more than N_0 items and the entropy of its position weight matrix is larger than $Entropy^0$, add it into M.

External cycle stops here.

End.

Assembly

After constructing the candidate collection, we should find the long repeat units from these fragments. If a fragment A is part of a repeat R, its neighbors may also be parts of R, so they should be also included in this candidate collection, and there should be $n_0 - 1$ positions in fragment A and one of its neighbors is the same. Therefore, we can assemble them by analyzing these candidate fragments. This is the assembly from the content of fragments, and we can use a simple exhaustive algorithm to realize it. In other words, for each fragment, we scan the whole set of fragments except that one. If the suffix of a fragment has enough bases that have the same prefix as another fragment, they would be assembled together. Using the numbers

of the same and different bases in the two fragments, we can calculate a quantity named as assemble score. This score can be used as a criterion to determine if an assembly can be executed.

There is also another method. If we know each position of the candidate fragments in the sequence, we can find the potential assembly relations by checking the positions. For example, if two fragments always occur at the similar position, and their related position is nearly fixed, they might belong to the same repeat unit. This is the assembly from the position of fragments. It also can be realized with a simple exhaustive algorithm.

Because assembling from position requires very huge computation (we have to find each fuzzy occurrence of each candidate fragment, which is time consuming. In practice, it is even worse. We usually have thousands of fragments after the fragment selection step), we use the following two steps as assembling policy. We first assemble the candidate fragments from their content, and obtain a small-scale composed fragment collection. Only the composed fragments whose length is larger than a threshold are reserved, and the others are discarded as noise fragments. After this step, there are usually tens of composed fragments reserved. Now, we can do assembling from position fragments and get the final repeat units.

Complexity analysis

In the fragment selection algorithm, the inner cycle would execute $L-n_0+1$ times, and in each cycle, we have to judge if the condition $B \in G$ is true. This is a time consuming operation. We construct a structure array with 4^{n_1} items, each item corresponding to a fragment of length n_1. Now, only small amount of computation is needed. In the following fragment filtration step, we do 4^{n_1} comparation at least, and calculate the entropy for the classes satisfying the dimension condition. Usually, 4^{n_1} is far less than $L-n_0+1$. That is to say, the computational complexity of the fragment selection algorithm is mainly brought by the inner cycle, $O(FL)$. It is a linear function of the input sequence length. Here F is the number of iteration times of inner cycle.

Just as we have motioned, the space complexity of this algorithm is another important factor to be considered. We use a structure array with 4^{n_1} items, each item includes a position weight matrix and a counter. The algorithm needs at least $4^{n_1}(4n_0-4n_1+1)$ integer units in memory. Compared with this number, the other memory usage is ignorable. Therefore, the space complexity of our projection algorithm is $O(4^{n_1})$. It is an exponential function of n_1, and this is the reason that we set a strict limit to n_1.

The complexity of the assembly step can be calculated as follows:

For the assembly from content, if there are W fragments in candidate collection, the computational complexity would be $O(W^2 n_0^2)$. When W and n_0 are small, it is acceptable. After this step, we keep the composed fragments with length larger than a threshold, and there are usually several dozen reserved fragments.

In the assembly from position, we first label all the positions of each composed fragments in the sequence using the fast fuzzy match search algorithm proposed by Myers (*12*). For a fragment of length k, the complexity of this algorithm is $O(kL)$. Here, L is the length of the input sequence. Therefore, the total complexity of the label step would be $O(KL)$. Here, K is the sum of all composed fragments. Then, we do assembling from position using exhaustive algorithm. If there are M composed fragments, the complexity of this operation would be $O(M^2 h)$. Here, h is the average times that each composed fragments occurred in the input sequence. Consequently, the total computational complexity of the assembling operation would be $O(KL+M^2 h)$, and its upper limit is the square of the length of the input sequence. In fact, the real test results show that the computing time is far less than this upper limit value, and usually, its complexity is ignorable compared with the fragment selection algorithm.

Results and Discussion

To test our algorithm all round, we first performed some tests with simulation data. We used the whole genome of *Escherichia coli* as the background sequence S, and inserted some artificial repeats with given parameters. The whole genome data was retrieved from the GenBank version 134.0 with accession number U00096. The total length is 4,639,221 bp, and the appearance probabilities of the four bases, A, C, G, and T, are 24.6%, 25.4%, 25.4%, and 24.6%, respectively. It is nearly a uniform distribution. We designed our test as follows: we first generate a sequence R with the given length as the repeat unit based on the background distribution. Then, we select m positions in sequence S and insert an instance

(noted as R_i) of the repeat R in each position. Here, instance R_i is formed after some independent mutations with probability p having been done on R. The in-del mutation may be or maynot be allowed, and, if in-del mutations are allowed, the three mutations should be occurred in equal probability.

During the tests, we selected several parameters. The results are given in Figure 2. To compare the results, we normalized the two sequences. That is, we first did a global alignment between the real repeat unit and the export results given by our algorithm. Then, we divided this score by the length of the real repeat unit, and the result would be between 0 and 1. This value was used as our normalized comparability score. In each test, we calculated the score for each output sequence, and used the highest one as the score of this run. Here we should point out that there is certain risk to use this score. For example, if no correct repeat unit is found, this score would be a random score, and it is meaningless. But on the other hand, the random score is not high enough (less than 0.5, for example). Hence, only when the score is higher than a reasonable threshold is this run meaningful. Therefore, our comparability was done under these conditions.

In the above tests, we used the following fixed parameters: $n_0 = 24$, $n_1 = 9$. In Figure 2A, the relations among the score, the repeat's occurrence number m and its length l are demonstrated. Here, mutation probability $p = 0.1$ and in-del mutations are permitted. As a contrast, Figure 2B gives the curves with the same parameters without in-del mutations. In Figure 2C, we fix the length of repeat unit as $l = 100$, and give the relation among score, m and p (in-del mutation is permitted). All results are the mean of over 20 runs, which were carried out on a computer with a 2.4 GHz CPU and 512 M memory. Each run was completed within 400 to 800 seconds.

From Figure 2A we can see that, when the mutation probability of the repeat unit is 0.1 and the occurrence number is 30, our algorithm can find them correctly in each run, with only slight errors at head and tail ends. The reason of these errors is as following: because we don't know the length of the repeat unit before they run, and only get the results by assembly, so some bases would be lost or added at the ends, and thus causes some trivial errors. Such errors usually bring more infection for the shorter repeats, that's why shorter l often has lower score in the figures.

Comparing Figures 2A and 2B, we find that with the same parameters, the scores with in-del errors are obviously lower than those without in-del errors. Further more, the score curves in Figure 2B rise much faster than those in Figure 2A and reach a very high level soon.

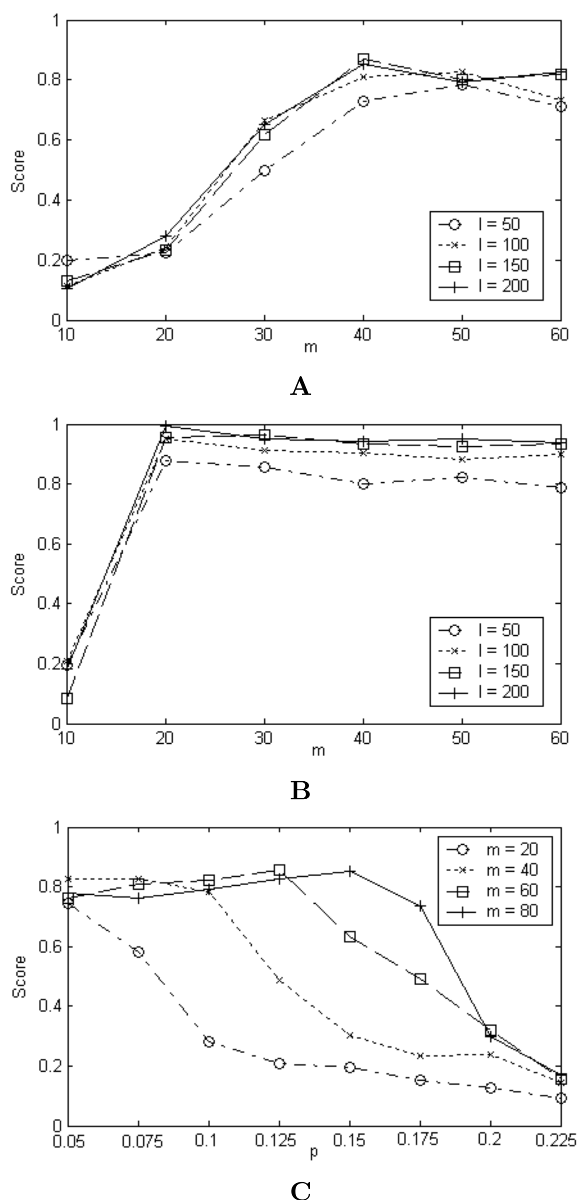

Fig. 2 The score curves of our algorithm. **A**. The score curves on repeat times, whose mutation probability is 0.1 with in-del errors; **B**. The same to figure 2A, only doesn't allow in-del errors; **C**. The score curve on mutation rate with in-del errors.

From Figure 2C we know that the mutation probability p plays an important role in the efficiency of our algorithm. When p rises to 20%, the scores usually reduce to 0.2 approximately, which is less than the threshold for random case. Our algorithm is effective when p is less than 10%. Besides, the score

curves have a rapid descending in a special position, which is related with the repeat time m: the larger the m, the larger the mutation probability. That is to say, the repeat time m can compensate the efficiency decrease brought by the rise of mutation probability in some degree.

We then ran our algorithm with the *E. coli* whole genome sequence. In the rude GenBank records, there are 482 elements noted as "repeat_region". After removing the redundant notes, the number of the real repeat region is 437. Most of the elements are repetitive extragenic palindromic element (REP element, 314 elements), and they locate in the forward chain. The REP element has the characters of the tandem repeats, and its length varies from several dozen base pairs to several thousand base pairs. It is difficult for our algorithm to find them. Except REP elements, the other repeat regions in *E. coli* genome are list in Table 1.

Table 1 Information of the Repeat Regions in *E. coli* Genome*

No.	Notation	Length (bp)	Occurrence		
			Total	Normal	Complement
1	IS1	768	7	3	4
2	IS2	1,331(1)	7	3	4
3	IS3	1,258(2)	5	3	2
4	IS5	1,195	11	9	2
5	IS30	1,221	4	3	1
6	IS186	1,343	3	3	0
7	IS Others	–	7	5	2
8	BoxC	56(3)	33	21	12
9	IRU	127(4)	19	8	11
10	Rhs	–(5)	5	5	0
11	Ter	–(6)	9	4	5
12	RSA	150/151	6	2	4
13	LDR	–(7)	4	4	0
14	Iap	–(8)	3	3	0

* In this table, Notation is the notation given in rude records. Length is the typical length of the repeat regions, and the flag "–" means there is no typical length for the repeat regions. Occurrence is listed in three columns, according to total number (Total), number in forward chain (Normal), and number in the converse-complement chain (Complement).

Note: (1) In the 7 instances of IS2, the length of one instance is 706 bp. (2) In the 5 instances of IS3, one instance has the length of 1,255 bp. (3) The length of instances of boxC varies from 51 to 59 bp. (4) Some instances of IRU have the length of 126 or 129 bp. (5) The 5 instances of Rhs are labeled as RhsA $\sim$ RhsE respectively, and their lengths are varied in region 5,000 $\sim$ 9,000 bp. (6) Two instances of Ter are labeled as Ter core, and their lengths are 11 bp. The other 7 instances are labeled as TerA $\sim$ TerG, their lengths are all 23 bp. (7) The 4 instances of LDR are labeled as LDR-a $\sim$ LDR-d, and their lengths varies from 200 to 600 bp. (8) The 3 instances of iap are labeled as iap(1), iap(7) and iap(14), with the length of 90, 395 and 823 bp, respectively.

From Table 1 we can find that, because many different sequences are involved, there is no obvious repeat fragments in the classes 7, 10, 11, 13, 14, and they are difficult to find. In the other repeat regions, the classes 1, 2, 3, 5, 6, 12 have few instances, and they are not easy to find, too. Compared with them, the classes 4, 8, 9 are easier to find.

We ran our algorithm with *E. coli* genome sequences. The parameters were set as follows: repeat unit length threshold as 50, repeat number threshold as 4, and mutation probability as 0.1. In the tests, we only searched the forward chain. The results are list in Table 2.

Table 2 shows that some instances of the three classes with stronger characters were found, although we can't find all instances of these classes. Our analysis proves that for the sequence with long typical length (IS5), we can find its all instances correctly. While for the classes with short typical length, we can only find its partial instances. This result demon-

Table 2 The Results of Our Algorithm for *E. coli* Genome Sequence*

No.	Notation	Reference information		Our result	
		Length (bp)	Occurrence	Length (bp)	Occurrence
1	IS5	1,195	9	1,202	9
2	BoxC	56	21	50	6
3	IRU	127	8	73	5
4	REP element	–	–	70	42
				102	15
5	Rhs	–	5	226	4
				175	5
				222	5
6	Unknown	–	–	5,200	5

* Here, Notation is the according label in rude GenBank records. Reference information is the information from the rude GenBank records, the last two columns are the results calculated by our algorithm.

Table 3 Information for the Repeat Sequence Found by Our Algorithm*

No.	Start position	Stop position	Length
1	223,620	228,890	5,271
2	3,939,280	3,944,459	5,180
3	4,032,969	4,038,231	5,263
4	4,164,087	4,169,351	5,265
5	4,205,574	4,210,753	5,180

* In this table, the lengths of the five instances are not exactly the same, and there are differences about 100 bp among them. Because the repeat region is longer than 5,000 bp, instead of listing each instance in the table, we only give the start and stop positions of these instances, and these positions are based on the whole genome sequence of *E. coli* from GenBank version 134.0.

strates that our algorithm could get a good efficiency for long repeat regions. On the other hand, our algorithm also found three segments of Rhs element, two segments of REP element, and an unknown long repeat region. Here we should emphasize that the length of the last unknown repeat region is more than 5,000 bp, and there are five instances in *E. coli* genome sequence. Our analysis indicates that the number of mismatch errors in these instances is less than 200 (the mutation probability is less than 4%). Therefore, from the sense of mathematics, it must be a repeat region, although it is not labeled in the rude GenBank records. The information of this repeat region is list in Table 3.

Conclusion

In this paper, we proposed a novel projection-assemble algorithm for the problem of long interspersed repeats finding. Our algorithm can find the repeats longer than a given length threshold and occurring more than the given occurrence threshold. For this problem, as far as we know, no efficient algorithm has been used widely. Our algorithm is an attempt in this subject, and the test results for both simulated and real biology data indicate that our algorithm is efficient and effective.

However, there are several fields in our algorithm that are worth for further research. First, we used the simple exhaustive algorithm in the assembly steps for the sake of the concision of the algorithm, with the cost of reducing the executed efficiency. Second, in our algorithm, we set n_1 as 9. For the sequence with length less than several dozen million base pairs, n_1 = 9 is enough, and the memory usage is reasonable. While for longer sequences, we should use larger n_1, which will use much more computer resources. The algorithm should be further improved. Third, this method can only be used in finding interspersed repeats, but not in finding tandem repeats. To find

tandem repeats, some feasible improved algorithms should use the position information more adequately in assembly, which requires more studies for the assembling algorithm with position. These are our future projects.

References

1. Lander, E.S., *et al.* 2001. Initial sequencing and analysis of the human genome. *Nature* 409: 860-921.
2. Brown, T.A. 1999. *Genomes*. BIOS Scientific Publishers, Ltd., Oxford, UK.
3. Benson, G. 1999. Tandem repeats finder: a program to analyze DNA sequences. *Nucleic Acids Res.* 27: 573-580.
4. Hauth, A.M. and Joseph, D.A. 2002. Beyond tandem repeats: complex pattern structures and distant regions of similarity. *Bioinformatics* 18: S31–37.
5. Kolpakov, R., *et al.* 2003. mreps: efficient and flexible detection of tandem repeats in DNA. *Nucleic Acids Res.* 31: 3672-3678.
6. Sonnhammer, E.L. and Durbin, R. 1995. A dot-matrix program with dynamic threshold control suited for genomic DNA and protein sequence analysis. *Gene* 167: GC1-10.
7. Kurtz, S., *et al.* 2001. REPuter: the manifold applications of repeat analysis on a genomic scale. *Nucleic Acids Res.* 29: 4633-4642.
8. Rigoutsos, I. and Floratos, A. 1998. Motif discovery without alignment or enumeration. In *Proceedings of the Second Annual International Conference on Computational Molecular Biology* (RECOMB), pp.221-227. ACM Press, New York, USA.
9. Hertz, G.Z. and Stormo, G.D. 1999. Identifying DNA and protein patterns with statistically significant alignments of multiple sequences. *Bioinformatics* 15: 563-577.
10. Pevzner, P. and Sze, S. 2000. Combinatorial approaches to finding subtle signals in DNA sequences. In *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology*, pp.269-278. AAAI Press, San Diego, USA.
11. Buhler, J. 2001. Search algorithms for biosequences using random projection. Ph.D. thesis. University of Washington, USA.
12. Myers, G. 1999. A fast bit-vector algorithm for approximate string matching based on dynamic programming. *J. ACM* 46: 395-415.