BIOINFORMATICS

Vol. 20 no. 4 2004, pages 460–466
DOI: 10.1093/bioinformatics/btg429

Rapid and sensitive dot-matrix methods for genome analysis

Yue Huang and Ling Zhang*

Lynnon Corporation, 116 rue du Milicien, Vaudreuil-Dorion, Quebec, Canada, J7V 9M4

Received on November 6, 2002; revised on April 23, 2003; accepted on July 22, 2003
Advance Access publication January 22, 2004

ABSTRACT

Motivation: Dot-matrix plots are widely used for similarity analysis of biological sequences. Many algorithms and computer software tools have been developed for this purpose. Though some of these tools have been reported to handle sequences of a few 100 kb, analysis of genome sequences with a length of >10 Mb on a microcomputer is still impractical due to long execution time and computer memory requirement.

Results: Two dot-matrix comparison methods have been developed for analysis of large sequences. The methods initially locate similarity regions between two sequences using a fast word search algorithm, followed with an explicit comparison on these regions. Since the initial screening removes most of random matches, the computing time is substantially reduced. The methods produce high quality dot-matrix plots with low background noise. Space requirements are linear, so the algorithms can be used for comparison of genome size sequences. Computing speed may be affected by highly repetitive sequence structures of eukaryote genomes. A dot-matrix plot of Yeast genome (12 Mb) with both strands was generated in 80 s with a 1 GHz personal computer.

Availability: The implementation of the described methods in C language is available at http://www.lynnon.com/dotplot/index.html

Contact: yhuang@lynnon.com

INTRODUCTION

Dot-matrix analysis is an efficient method to search for similarities between two sequences (Gibbs and McIntyre, 1970; Argos, 1987; Risler *et al*., 1988; States and Boguski, 1991). It is often used to find insertions or deletions, direct or inverted repeats in protein and DNA sequences, and to predict regions in RNA that might be self complementary and form a double-stranded region or secondary structure (Maizel and Lenk, 1981; Dumas and Ninio, 1982). In the dot-matrix plot, all possible matches of residues between two sequences are represented graphically, allowing users to identify the most significant matches. While a simple display of sequence alignment is used to compare two or more short sequences, a graphic presentation is more useful for analyzing long sequences such as genomic DNA. A limitation of dot-matrix methods is that the sequences of the actual regions are not shown on a plot. However, with recent advance in computing technology and graphical interface, this limitation is less relevant. A dot-matrix plot can be displayed on the computer terminal, and software tools are available to provide an interactive environment so that many types of analysis may be performed (Rechid *et al*., 1989; Sonnhammer and Durbin, 1996). For instance, an interactive computer program may allow investigators to visualize the sequence alignment of the regions of interest on a dot-matrix plot, and identify functional groups using annotation information.

In dot-matrix plots, long lines show similarity regions between two sequences, while short dots may represent random matches or background noises. Visualization of matching regions can be improved by filtering out random matches using a threshold window. Filtering is achieved by sliding the window over the plot and disqualifying matches shorter than the window. Such filtration is computationally expensive and not practical for long sequences. Dotter (Sonnhammer and Durbin, 1996) is a widely used dotplot program that computes sequence similarity and displays a grayscale image. Although it is fast and accurate in plotting short sequences, generating dotplots on a microcomputer for sequences longer than 1 Mb is extremely slow. Many algorithms have been developed for improving the speed of sequence comparison. Fristensky (1986) suggested using oligomer tables containing indexes to the positions of occurrence of a list of oligomers. This method may greatly improve the speed of search for identical residues, but the sensitivity decreases and noise increases when computing long sequences. Dottup from EMBOSS (Rice *et al*., 2000) uses a word search algorithm to look for exact matches between two sequences. It is fast with short sequences, but not especially sensitive for creating dot-matrix plots. It is only an acceptable method for displaying regions of substantial similarity. Lefevre and Ikeda (1994) developed a fast word search algorithm using position end-set tree structure. Such method improved computing speed and also allowed imperfect matches in similarity search.

The new methods we describe here use a fast search algorithm to identify short similar sequence regions in the first step of comparison. The algorithm employs a lookup

*To whom correspondence should be addressed.

table that contains all possible combinations of a word. Similar methods have been used in repetitive sequence search (Dumas and Ninio, 1982), sequence alignment (Higgins and Sharp, 1988) and similarity searches in database (Wilbur and Lipman, 1983; Lipman and Pearson, 1985; Higgins and Stoehr, 1992). The search methods are very fast, however, they cannot be directly used for dot-matrix plot of lengthy sequences. Our methods make the word search algorithm suitable for dot-matrix analyses of genomes.

METHODS

Our objective is to search for similar regions of two nucleotide sequences N1 and N2, of lengths L1 and L2, respectively. We shall here describe two computing methods to locate these regions. In both methods, a fast search algorithm is used to find out short identical or similar units with length k between the two sequences. The sequence region subsequent to each unit is then filtrated using a threshold score s. A dot is produced where the two sequences match with a length of w or greater.

Method 1: dot-matrix plot by direct comparison of two sequences

In a first step, this method identifies short identical sequence regions between N1 and N2 using the fast search algorithm that employs a lookup table of all possible sequences. Prior to comparison, we build a location table of all k-tuples for sequence N1 using the algorithm described by Dumas and Ninio (1982). If the sequence is made from p elements in each position ($p = 4$ for DNA, $p = 22$ for protein), there are p^k possible different k-tuples. Any k-tuple can be converted into an integer between 1 and p^k. Thus, a one-dimensional array, C, of length p^k is used to point positions of all k-tuples in N1. The content of C is initially set to nil. Another one-dimensional array, D, of length of $\mathrm{L1} - k$, is used as position pointers. $D[i], i = 1, 2 \ldots, \mathrm{L1} - k$, is initially set to the coded form, $c(i)$, of each k-tuple of N1. To start table construction, the first position, 1, is assigned to $C[c(1)]$ and $D[1]$ is set to nil. In a single pass through N1, each position i is assigned to $C[c(i)]$. When the k-tuple $c(i)$ is found in another location, position j, the previous position, $C[c(i)]$, is assigned to $D[j]$, and j is assigned to $C[c(i)]$. When the table is constructed, $C[c(i)]$ points to the last occurrence of the k-tuple $c(i)$ in N1. For instance, by looking at $C[c(1)]$ and following the pointer in array D, one can find all locations of the k-tuple of the first position, $c(1)$. The following example shows the construction of C and D arrays with $k = 2$.

To start comparison, the k-tuple of position $i, i = 1, 2, \ldots, \mathrm{L2} - k$, in N2 is coded to $d(i)$ using the same coding method. The last occurrence of the k-tuple in N1 is found at position $C[d(i)]$. Following the pointer of in array D, all positions of the k-tuple can be identified in N1. In such a way, N2 is compared with N1 through all individual k-tuples.

In the initial step, we have identified each pair of k-tuple between N1 and N2. In the second step of this method, an explicit comparison is performed to qualify a k-tuple as a real match region. A score threshold may be used in this step. A match between two nucleotides is given a score of 0, and -1 is assigned to a mismatch. A real match region is defined as the following: for a window size w, where $w > k$, the score is not less than threshold s, where s is a negative number. When a k-tuple match is found, the subsequent regions between N1 and N2 are compared using the qualification criteria. Such comparison ends when the score drops below s. With a match length longer than w, the region is recorded as a real match. A dot-matrix plot consists of all the records of real matches. The following pseudocode shows the calculation of a match length above the threshold s.

```
MatchLengthAboveThreshold(p,q,s)

Integer   p              // k-tuple position of N1
          q              // k-tuple position of N2
          s              // Threshold score
Integer   MatchLength    // Match Length
                            between N1 and N2
Pointer   RingBuffer     // A ring buffer
                            to record score

SetToZero(RingBuffer);
                         // Set all positions of RingBuffer
                            to 0
MatchLength = k;
                         // Initiate match length
While Sum(RingBuffer) > s do
{
   MatchLength = MatchLength + 1;
                         // Move position further
   RingBufferGetScore(N1[p + MatchLength],
                         N2[q + MatchLength]);
}
return MatchLength;
```

	A	G	C	T	C	G	A	T	C	G	A	G	T	C	T	C	G	A	G	T	A	G
Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
D: initial set	3	10	8	14	7	9	4	14	7	9	3	12	14	8	14	7	9	3	12	13	3	
D: final value	0	0	0	0	0	0	0	4	5	6	1	0	8	3	13	9	10	11	12	0	18	
C:	0	0	21	7	0	0	16	14	17	2	0	19	20	15	0	0						
k-tuple	AA	AC	AG	AT	CA	CC	CG	CT	GA	GC	GG	GT	TA	TC	TG	TT						

Method 2: dot-matrix plot by comparison with converted sequences

The first method does not allow mismatch during initial stage when the lookup table is built. Some similar regions may be missed due to lack of a segment of perfect matches. This method can be improved by introducing limited mismatches using converted sequence. A biologically significant conversion is the translation of a nucleotide sequence to the corresponding amino acid sequence. In this method, the two nucleotide sequences, N1 and N2, are first converted into amino acid sequences using Universal or a user-defined table. The three reading frames of N1 are translated to sequence A11, A12, A13 and N2 to A21, A22, A23. In a first step, short identical amino acid sequences, *k*-tuples, can be identified in each pair sequences, (A11, A21), (A11, A22), (A11, A23), (A12, A21),. . . , (A13, A23), using the fast search algorithm described in Method 1.

The second step of comparison is performed between the nucleotide sequences, N1 and N2. The positions of each *k*-tuple match of amino acid sequence are mapped to N1 and N2. A score is computed on the initial matches and subsequent nucleotides between N1 and N2. When scoring the initial matches between N1 and N2, mismatches are ignored due to the perfect match between amino acid sequences. Any subsequent mismatch may decrease the score. The scoring is stopped where the score falls below the threshold *s*. A region is qualified as a real match and recorded if the length is not shorter than *w*.

The following pseudocode illustrates the algorithm of Method 2.

```
Pointers   N1,      // pointer to nucleotide sequence 1
           N2,      // pointer to nucleotide sequence 2
           A1[3],   // pointer to list of amino acid
                       sequences from N1
           A2[3],   // pointer to list of amino acid
                       sequences from N2
           C1[3],   // pointer to k-tuple table for A1
           C2[3],   // pointer to k-tuple  table for A2
           D1[3],   // pointer to position table for A1
           D2[3],   // pointer to position table for A2
Integers   L1,      // length of N1
           L2,      // length of N2
           AL1,     // length of A1
           AL2,     // length of A2
           p,       // temporary position in A1
                       when passing C1 and D1
           q,       // temporary position in A2
                       when passing C2 and D2
           w,       // minimum length to
                       qualify a dot
           s,       // Threshold score of
                       similarity between
                       two sequences
           ml       // Temporary match length
                       at position point (p,q)

ConvertSequence(N1, A1, L1);     // Translate nucleotides
                                    to amino acid sequence

ConvertSequence(N2, A2, L2);     // Translate nucleotides
                                    to amino acid sequence
For i=1 to 3 do
{
  BuildLookupTable(A1[i],C1[i],D1[i],AL1);
     // Build lookup table for A1[i]
  BuildLookupTable(A2[i],C2[i],D2[i],AL2);
     // Build lookup table for A2[i]
}
For i = 1 to 3 do
{
   p = C1[i][1];
   while p > 0 do
   {
     for j = 1 to 3 do
     {
       q = C2[j][1]
       while q > 0 do
       {
         if TheKtupIsValid(p,q) then    // Compute only the
         {      // ktup pair not covered by the one in previous
                // position
           ml = MatchLengthAboveThreshold(p,q,s)
           if ml > w then RecordDot(p,q,ml)
         }
         q = D2[j][q]
       }
     }
     p = D1[i][p]
   }
}
```

The ConvertSequence module can be the translation of DNA to amino acid sequence, or other conversions. Nucleotide sequence is translated to amino acids in three reading frames. The other three reading frames on the antisense strand are also compared when computing the dotplot for the reverse strand.

Another effective conversion is the Two for Three (TFT) method, which is similar to amino acid translation. In TFT, a DNA sequence is grouped into triplets. We introduce a code number for each triplet. The code of a triplet is calculated using the first two bases, instead of translated to amino acid. Hence, a genetic table of 64 triplets can be converted into 16, not 20 codons.

Space and time

To construct search tables for N1 and N2, the algorithm requires $O(L1 + L2)$ space in addition to a fixed amount

of space p^k for k-tuple table. The space requirement can be reduced to $O(\mathrm{L1})$ when N2 sequence length is small or similar to the size of p^k. In this case, construction of search table for N2 is not necessary since it does not improve speed or sensitivity. Typically, 5–10 MB of memory are required for each megabase of sequence.

The time requirement of the algorithms can be subdivided into Table Construction, k-tuple Search and Similarity Examination. Table Construction requires $O(\mathrm{L1} + \mathrm{L2})$ time. In k-tuple Search, the speed depends on the occurrence of a particular k-tuple on both sequences. If it happens N times in one sequence and M in the other, there are $N \times M$ matches to search. Thus it requires $O(N * M)$ time. The speed of Similarity Examination depends on the similarity between the two sequences following each k-tuple pair. Low similarity requires little examination time. High similarity regions result in long segments to compute and consequently require more time. In general, k-tuple Search is the time limiting factor of the methods. Therefore, the time requirement is not linear, nor $O(\mathrm{L1} * \mathrm{L2})$.

Sensitivity

The similarity examination in our algorithms uses an explicit comparison method which is highly sensitive. Prior to examination, every similarity region must be firstly located by k-tuple search. Thus, the sensitivity depends on the existence of k-tuple pairs. In Method 1, a k-tuple pair is usually an identical sequence of 6–14 bp. In Method 2, a k-tuple pair is usually a fragment of similar sequence of 10–21 bp. The algorithms may locate all homologous segments that are led by a k-tuple pair, and they will fail to identify a similarity region without a k-tuple pair leading that region.

Prokaryote genomes, several megabases long, usually have few repetitive structures and no introns. Method 2 can be significantly more sensitive than Method 1, since identical bases of k-tuple between two genomes are not required and coding regions are easy to be detected. Higher eukaryote genomes, more than 100 MB long, are full of repetitive structures and introns which do not follow the rule of translation. Thus, Method 2 using translation conversion may not have much impact on the sensitivity of the analysis of high eukaryote genomes.

EXAMPLES

The algorithm was implemented using C language and a program was compiled for the MS Windows operation system. Dot-matrix analysis was performed on Windows XP system with a personal computer equipped with a 1 GHz AMD processor and 640 MB of memory.

Genomic sequences of *Thermoplasma volcanium* and *Thermoplasma acidophilum*

The genomic sequences of *T.volcanium* (GenBank accession no. BA000011), 1 584 804 bp, and *T.acidophilum* (GenBank accession no. AL139299), 1 564 906 bp, were retrieved from the web site of the National Center for Biotechnology Information, http://www.ncbi.nlm.nih.gov. The two sequences were compared using the described dot-matrix methods on both strands of DNA. Figure 1 shows the dot matrix generated using Method 2 with a window size of 27 and with no mismatches allowed. A k-tuple of 5 was used and the

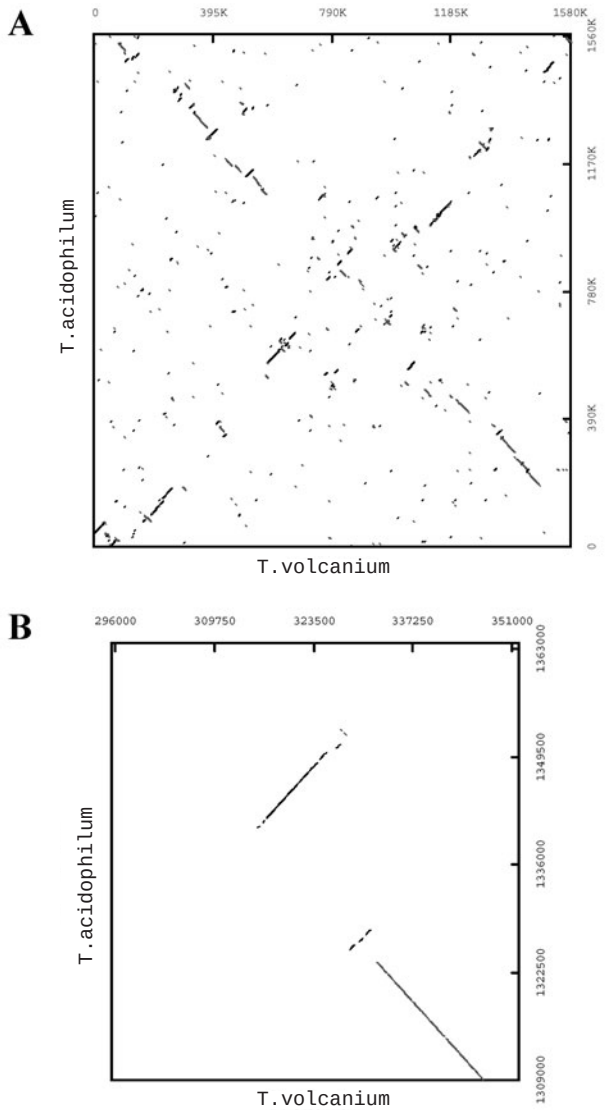

Fig. 1. Dot-matrix plot of the *T.volcanium* and *T.acidophilum genomes*. The dotplot was computed with Method 2 using perfect matches of amino acid sequence. Similarity search was performed with k-tuple = 5, and a window size = 27 nt. The comparison of the two sequence strands took 15.6 s. The dots towards northeast represent the comparison of two sense sequences and the dots towards southeast, the comparison between one sense sequence and one antisense sequence. (**A**) The entire plot of genomes of *T.volcanium* and *T.acidophilum*. (**B**) A zoomed region of plot A, *T.volcanium* from 296 000 to 346 000 and *T.acidophilum* from 1 313 000 to 1 361 000.

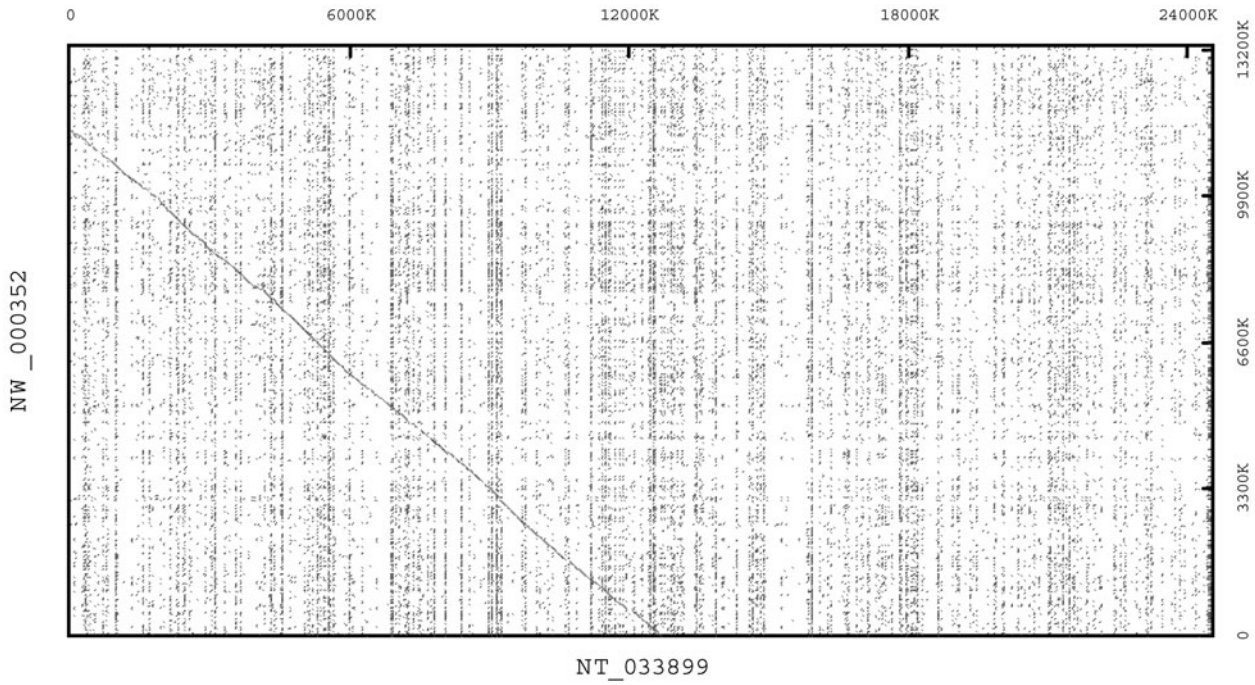

Fig. 2. Dot-matrix plot of DNA sequences from human chromosome 11 and mouse chromosome 9. A contig of mouse chromosome 9 (NW_000352) was compared with human chromosome 11 (NT_033899). The dotplot was computed using Method 1. The analysis was performed with a window size of 69 and a mismatch of 10. A *k*-tuple of 12 was used and the computation lasted 8 min. The dots towards northeast represent the comparison of the two sense sequences and the dots towards southeast, the comparison of the sense sequence of NT_033899 and the antisense of NW_000352. A strong homologous region is observed, between 1–12.7 Mb of NT_033899 and 1–11.4 Mb of antisense NW_000352.

computation took 15.6 s, including the table construction for both strands and the search for similarity. The dots towards the northeast represent the comparison of the two sense sequences and the dots towards the southeast, the comparison of one sense sequence with the other antisense sequence.

Based on sequence similarity, *T.volcanium* and *T.acidophilum* are two closely related Archaea microbes (Boucher *et al*., 2001; Slesarev *et al*., 2002). The dot matrix shows a clear picture of gene organization of both genomes. Most of sequence segments of one genome can be found on the other one, but the genes are organized in different orders and directions.

The two genome sequences were also analyzed using Method 1. A dotplot was produced using a window size of 27 with four allowed mismatches. A *k*-tuple of 11 was used and the computation lasted 4.8 s. Although the dot matrix was similar to that generated with Method 2, detection of weak homologous regions was less sensitive.

Genome contigs of human chromosome 11 to mouse chromosome 9

In order to demonstrate the ability of comparing large genome sequences, we have computed the dot matrix between a contig of the human chromosome 11 (GenBank accession no. NT_033899), 24 547 311 bp, and a contig of the mouse chromosome 9 (GenBank accession no. NW_000352), 13 254 964 bp.

Figure 2 shows a dotplot generated by Method 1. The analysis was performed with a window size of 69 and a mismatch of 10. A *k*-tuple of 12 was used and the computation lasted 8 min. The dots towards the northeast represent the comparison of the two sense sequences and the dots towards the southeast the comparison of the sense sequence of NT_033899 with the antisense of NW_000352. As observed, a strong homologous region crosses the plot, between 1 to 12.7 Mb of NT_033899 and 1 to 11.4 Mb of antisense NW_000352. Although the two genomes were organized almost identically in this region, there were 1.3 Mb, or ~10%, extra nucleotides in the human genome. The human genome also contained more repetitive sequences as indicated by vertical line pattern in the plot.

When the two genomes were analyzed using Method 2, execution time was twice longer than with Method 1. Using Method 2 did not improve the sensitivity of the dotplot of the two genomes.

DISCUSSION

Although many dot-matrix tools for genomic DNA sequence analysis have been developed, the challenges remain the

same: sensitivity of similarity detection, execution time and memory requirement. A 'perfect' dot-matrix algorithm would deliver high sensitivity for similarity detection with short execution time and low memory requirement. Previous developed algorithms (Sonnhammer and Durbin, 1996; Rice *et al*., 2000; Lefevre and Ikeda, 1994; Staden, 1982; Reich and Meiske, 1987) often require a long execution time due to background noises or random matches of lengthy genomic sequences. Our methods are designed to avoid computing random matches.

Comparison with other algorithms

We have compared the algorithms with Dotter (Sonnhammer and Durbin, 1996), a widely used dotplot program. Dotter uses an explicit comparison method to compute similarity of two sequences from one end to the other. Thus, it shows the highest sensitivity possible. It uses very small amount of memory and requires $O(\mathrm{L1})$ space. Dotter stores the dots in a $\mathrm{L1} \times \mathrm{L2}$ matrix. A compressed matrix is used when computing large sequences, thus sensitivity may be decreased for large genomes. Dotter requires $O(\mathrm{L1} * \mathrm{L2})$ time for computing for a dotplot. On a 700 MHz computer, Dotter takes 155 s to compare 25 kb of *T.volcanium* and *T.acidophilum* and 622 s to compare 50 kb. Thus, it would take ~612 000 s, or 7 days to compare the entire genomes. With our algorithms, Method 1 takes 2.9 s and Method 2 takes 4.5 s. In terms of sensitivity, we have compared a few published dotplots produced by Dotter with our methods. No significant loss of sensitivity was observed with our methods. For instance, we have produced a dot matrix between *rice* BAC AP003412 and *Arabidopsis* BAC AC002510 and all elements indicated in the original publication (Salse *et al*., 2002) are well identified in our plot. When comparing BAC 635P2 with BAC 36I5, Method 1 clearly located all transposable elements (Dubcovsky *et al*., 2001).

Dottup of EMBOSS package (Rice *et al*., 2000) also produces dotplots of DNA and protein sequences. Dottup uses a word search method to accelerate the similarity search and is very fast with short sequences. Since it computes only perfect matches, its sensitivity is low. We have tested Dottup on a 700 MHz G4 computer with MacOSX and 384 MB RAM. It took 5, 18, 510 and 6300 s to compute dotplots of *T.volcanium* and *T.acidophilum* with the first 50, 100, 500 kb and 1.5 Mb (whole genome), respectively. The memory requirement was 5, 11, 54 and 165 MB for four size sequences, respectively. On the same computer, our Method 1 took 0.3 s for the sequences of 100 kb and 3 s for the entire genome sequences and requires 1 and 15 MB of memory, respectively.

Method optimization for highly repetitive sequences

The algorithms requires $O(N * M)$ time for screening similarity regions, where N and M are the occurrences for each k-tuple on both sequences. N and M are normally very small numbers. For instance, a random sequence of 12 bases happens in theory only once in 4^{12}, or 16.8 MB. Genomes of prokaryotic organisms have very few structural repetitive units, therefore, the algorithms run very fast with prokaryotic genomes.

Eukaryotic genomes contain highly repetitive segments as a part of the genome organization, producing dotplots with vertical or horizontal line patterns. The repetitive nature of eukaryotic genomes may significantly increase N and M. In some case, this may not significantly affect the speed of computation. For example, it takes 80 s using Method 1 to compute the dotplot of Yeast genome to itself for both strands. Although AAAAAAAAAAAA repeats 2169 times and TTTTTTTTTTTT 2203 times, removing these repeats does not affect significantly computing time. In other cases, these repetitive sequences may constitute a significant percentage of all searching positions and computations lay heavily on the repetitive regions. The algorithms may be modified to provide a way to avoid computing such k-tuples. One of improvements for eukaryotic genomes is to ignore the k-tuples of repetitive segments. We may set a maximum of occurrences for each k-tuple, mk. During the construction of arrays C and D, any k-tuple that occurs more than mk times would be set to nil. Thus, these positions will not be searched. This modification may result in a slightly lower sensitivity since those similarity regions led by such k-tuples would not be detected. However, our tests have not shown any significant decrease in sensitivity.

The comparison of the contigs of human chromosome 11 and mouse chromosome 9, Method 1 took 8 min to compute. In the 24.7 Mb of chromosome 11, the k-tuple AAAAAAAAAAAA occurred 15 561 times and TTTTTTTTTTTT 16 064 times. When the two k-tuples were ignored in the similarity search, the computation time was reduced by 25% . If the repetitive structure is not of the interest and 16 of the most frequent repeats were ignored, the computation time was further reduced by 50% . In both cases, no visible sensitivity decrease in the detection of overall homology was observed. Removing highly repetitive sequences may be especially important for self-comparison of eukaryotic genomes. For instance, if chromosome 11 is compared with itself, the AAAAAAAAAAAA k-tuple will produce 242 millions computation loops and the TTTTTTTTTTTT k-tuple 258 millions. These loops may significantly reduce computing speed.

ACKNOWLEDGEMENTS

The authors thank Dr Urs Kuhnlein for correction of the manuscript, and the referees' comments and suggestions that substantially improved the presentation of this paper.

REFERENCES

Argos,P. (1987) A sensitive procedure to compare amino acid sequences. *J. Mol. Biol.*, **193**, 385–396.

Boucher,Y., Huber,H., L'Haridon,S., Stetter,K.O. and Doolittle,W.F. (2001) Bacterial origin for the isoprenoid biosynthesis enzyme HMG-CoA reductase of the Archaeal orders Thermoplasmatales and Archaeoglobales. *Mol. Biol. Evol.*, **18**, 1378–1388.

Dubcovsky,J., Ramakrishna,W., SanMiguel,P.J., Busso,C.S., Yan,L., Shiloff,B.A. and Bennetzen,J.L. (2001) Comparative sequence analysis of colinear barley and rice bacterial artificial chromosomes 1. *Plant Physiol.*, **125**, 1342–1353.

Dumas,J.P. and Ninio,J. (1982) Efficient algorithms for folding and comparing nucleic acid sequences. *Nucleic Acids Res.*, **10**, 197–206.

Fristensky,B. (1986) Improving the efficiency of dot-matrix similarity searches through use of an oligomer table. *Nucleic acids Res.*, **14**, 597–610.

Gibbs,A.J. and McIntyre,G.A. (1970) The diagram, a method for comparing sequences. Its use with amino acid and nuceotide sequences. *Eur. J. Biochem.*, **16**, 1–11.

Higgins,D.G. and Sharp,P.M. (1988) CLUSTAL: a package for performing multiple sequence alignment on a microcomputer. *Gene*, **73**, 237–244.

Higgins,D.G. and Stoehr,P. (1992) EMBLSCAN: fast approximate DNA database searches on compact disc. *Comput. Appl. Biosci.*, **8**, 137–139.

Lefevre,C. and Ikeda,J.E. (1994) A fast word search algorithm for the representation of sequence similarity in genomic DNA. *Nucleic Acids Res.*, **22**, 404–411.

Lipman,D.J. and Pearson,W.R. (1985) Rapid and sensitive protein similarity searches. *Science*, **227**, 1435–1441.

Maizel,J.V. and Lenk,R.P. (1981) Enhanced graphic matrix analysis of nucleic acid and protein sequences. *Proc. Natl Acad. Sci., USA*, **78**, 7665–7669.

Rechid,R., Vingron,M. and Argos,P. (1989) A new interactive protein sequence alignment program and comparison of its results with widely used algorithms. *Comp. Appl. Biosci.*, **2**, 107–113.

Reich,J.G. and Meiske,W. (1987) A simple statistical significance test of window scores in large dot matrices obtained from protein or nucleic-acid sequences. *Comp. Appl. Biosci.*, **3**, 25–30.

Rice,P., Longden,I. and Bleasby,A. (2000). EMBOSS: the European Molecular Biology Open Software Suite. *Trends Genet.*, **16**, 276–277.

Risler,J.L., Delorme,M.O., Delacroix,H. and Henaut,A. (1988) Amino acid substitutions in structurally related proteins, a pattern recognition approach. Determination of a new and efficient scoring matrix. *J. Mol. Biol.*, **204**, 1019–1029.

Salse,J., Piégu,B., Cooke,R. and Delseny,M. (2002) Synteny between *Arabidopsis thaliana* and rice at the genome level: a tool to identify conservation in the ongoing rice genome sequencing project. *Nucleic Acids Res.*, **30**, 2316–2328.

Slesarev,A.I., Mezhevaya,K.V., Makarova,K.S., Polushin,N.N., Shcherbinina,O.V., Shakhova,V.V., Belova,G.I., Aravind,L., Natale,D.A., Rogozin,I.B. *et al*. (2002) The complete genome of hyperthermophile *Methanopyrus kandleri* AV19 and monophyly of archaeal methanogens. *Proc. Natl Acad. Sci., USA*, **99**, 4644–4649.

Sonnhammer,E.L.L. and Durbin,R. (1996) A dot-matrix program with dynamic threshold control suited for genomic DNA and protein sequence analysis. *Gene*, **167**, GC1–GC10.

Staden,R. (1982) An interactive graphics program for comparing and aligning nucleic acid and amino acid sequences. *Nucleic Acids Res.*, **10**, 2951–2961.

States,D.J. and Boguski,M.S. (1991) Similarity and homology. In *Sequence Analysis Primer*. Stockton Press, New York, pp. 92–124.

Wilbur,W.J. and Lipman,D.J. (1983) Rapid similarity searches of nucleic acid and protein data banks. *Proc. Natl Acad. Sci., USA*, **80**, 726–730.