ELSEVIER Gene 167 (1996) GC1–10

GENE-COMBIS

A dot-matrix program with dynamic threshold control suited for genomic DNA and protein sequence analysis

Erik L.L. Sonnhammer [a,*], Richard Durbin [a,b,1]

[a] *Sanger Centre, Hinxton Hall, Cambridge CB10 1RQ, UK*
[b] *MRC Laboratory of Molecular Biology, Hills Road, Cambridge CB2 2QH, UK*

Received 30 June 1995; revised 17 September 1995; accepted 4 October 1995
Published 17 October 1995

Abstract

Graphical dot-matrix plots can provide the most complete and detailed comparison of two sequences. Presented here is **DOTTER** [2], a dot-plot program for X-windows which can compare DNA or protein sequences, and also DNA versus protein.

The main novel feature of DOTTER is that the user can vary the stringency cutoffs interactively, so that the dot-matrix only needs to be calculated once. This is possible thanks to a ‘Greyramp tool’ that was developed to change the displayed stringency of the matrix by dynamically changing the greyscale rendering of the dots. The Greyramp tool allows the user to interactively change the lower and upper score limit for the greyscale rendering. This allows exploration of the separation between signal and noise, and fine-grained visualisation of different score levels in the dot-matrix.

Other useful features are dot-matrix compression, mouse-controlled zooming, sequence alignment display and saving/loading of dot-matrices. Since the matrix only has to be calculated once and since the algorithm is fast and linear in space, DOTTER is practical to use even for sequences as long as cosmids.

DOTTER was integrated in the gene-modelling module of the genomic database system **ACEDB** [3]. This was done via the homology viewer BLIXEM in a way that also allows segments from the BLAST suite of searching programs to be superimposed on top of the full dot-matrix. This feature can also be used for very quick finding of the strongest matches. As examples, we analyse a *Caenorhabditis elegans* cosmid with several tandem repeat families, and illustrate how DOTTER can improve gene modelling.

Keywords: Sequence analysis, DNA; Genome analysis; User interface; Analysis; Software, package; Laboratory informatics; Community informatics

1. Introduction

Ever since the introduction of graphical dot-matrix plots to sequence analysis [1,2], they have been among the most popular methods for analysing similarity between two sequences, particularly for gaining a good picture of the similarity between repeated domains.

The original dot-plot concept of drawing one sequence along the horizontal axis and the other along the vertical axis of a coordinate system, and drawing a dot where two residues match has essentially stayed the same. Regions of similarity between the sequences will result in a diagonal row of dots, whereas spurious matches give rise to a background of single dots. A standard filtering technique to reduce the noise is to apply a window along the diagonals and only draw a dot in the centre of the window if the sum of all dots in the window exceeds some score threshold.

One problem is that the optimal threshold for drawing a dot is hard to guess a priori. Poor choice of threshold may result in dot-plots either too full of noise or lacking the relevant diagonals. This can be frustrating, since changing the threshold usually requires recalculation of the entire dot-plot, which often is very time consuming. Estimating the threshold by probabilistic methods [3–7] can be of use for finding the approximate border region between signal and noise, but still usually requires recalculation of the dot-matrix at different score levels. Inspecting the dot-plot at different thresholds is very informative since it gives a

Abbreviations: MSP: Maximal Segment Pair
[*] Corresponding author. Fax: +44 1223 494919; Tel.: +44 1223 494991. E-mail: esr@sanger.ac.uk
[1] Fax: +44 1223 494978. E-mail: rd@sanger.ac.uk
[2] http://www.sanger.ac.uk/dotter.html
[3] ftp://ftp.sanger.ac.uk/pub/acedb

better picture of the strength of a diagonal relative to the noise and other [8]. **DOTTER** [4] was created to make this interactive aspect of dot-plots more powerful than in previous implementations.

Improvements on the classical single-bit dot-plot (where dots are either on or off) have been to encode the score of a dot by colours [9–11] or by lines of varying thickness [7]. However, none of these programs can plot more than 16 different colours or shapes, and since they can not be modified dynamically to other thresholds, they do not eliminate the need for recalculation if another stringency rendering is required.

Modern graphics hardware offers new possibilities for addressing this problem. DOTTER allows the user to set score thresholds dynamically *after* the dot-matrix has been calculated, using the X-windows system for changing screen colours on 8-bit displays. This is done by a mouse-controlled ‘Greyramp’ tool which lets the user modify two score thresholds which can either be used as a strict - all or nothing - cutoff, or as a smooth rendering of many different score levels at once. Dots scoring below the first threshold are invisible and dots scoring above the second threshold get the maximum intensity while dots scoring between the thresholds are rendered with an intensity proportional to their score. Employing 128 different greyscale colours ensures a smooth range of intensity values.

Computationally, the main problem with dot-plots is that the execution time is proportional to the product of the lengths of the sequences, which makes long sequences very time consuming. This problem has been attacked by heuristic approaches [12,13] and trees combined with heuristics [14]. Such techniques can give improvements in speed of several orders of magnitude, at the cost of generating a not entirely correct dot-matrix. For long sequences, where an overview of the strongest matches is of main interest, such approximations may be acceptable, but for detailed analysis of weak similarities the full matrix still needs to be calculated. We recognise the usefulness of such fast methods and have therefore equipped DOTTER with the ability to also read in matches produced by the BLAST suite [15]. Displaying ungapped matches from BLAST is also informative since it shows the extent of high-scoring segments.

DOTTER is a versatile tool for dot-matrix comparisons of DNA and protein sequences. It can produce dot-plots for DNA vs. DNA, protein vs. protein, and DNA vs. Protein. For DNA, it can draw the reverse complement diagonals in the same dot-matrix as the forward ones. For DNA vs. protein, it translates the DNA sequence in the three forward frames and draws them all in the same dot-matrix. All modes feature tools to inspect the sequence alignment of any diagonal.

[4] http://www.sanger.ac.uk/dotter.html

2. Materials and methods

DOTTER [5] was written in the ANSI C language, using the graphics routines from the **ACEDB** [6] graphics library [16]. Supported platforms are UNIX X-windows workstations from Silicon Graphics, Digital and SUN.

2.1. Generating the dot-matrix

Here, the dot-matrix will not simply contain a zero or a one (one bit) for each dot as in the traditional dot-plot, but a value between 0 and 255 (8 bits $=$ one byte). The dot-matrix thus contains scores, averaged over a chosen window-span, but we prefer not to call it a ‘score matrix’ to avoid confusion with the well-known pairwise score matrices, such as PAM120 and BLOSUM62. The dot-matrix needs only to be calculated once for a given window-span.

For maximum speed, we precalculate score vectors for every possible symbol in the vertical sequence along the horizontal sequence [17]. For DNA, this requires $4+1$ score vectors (1 extra for unknown symbols), and for protein $20+2+1$ (20 amino acids, 2 for ambiguity symbols and 1 for unknowns). This makes execution faster since the few score vectors only have to be calculated once and are later added to and removed from the sliding window-sums. The window-sums are calculated for consecutive windows along the diagonal in a sliding manner by simply adding the next score and subtracting the last score inside the window. Instead of calculating the window-sums for one diagonal at a time however, we keep a horizontal vector of all window-sums and add and subtract the precalculated score vectors row by row.

The following pseudocode outlines the algorithm. The score vectors are assumed to be initialised with the scores from the pairwise score matrix used.

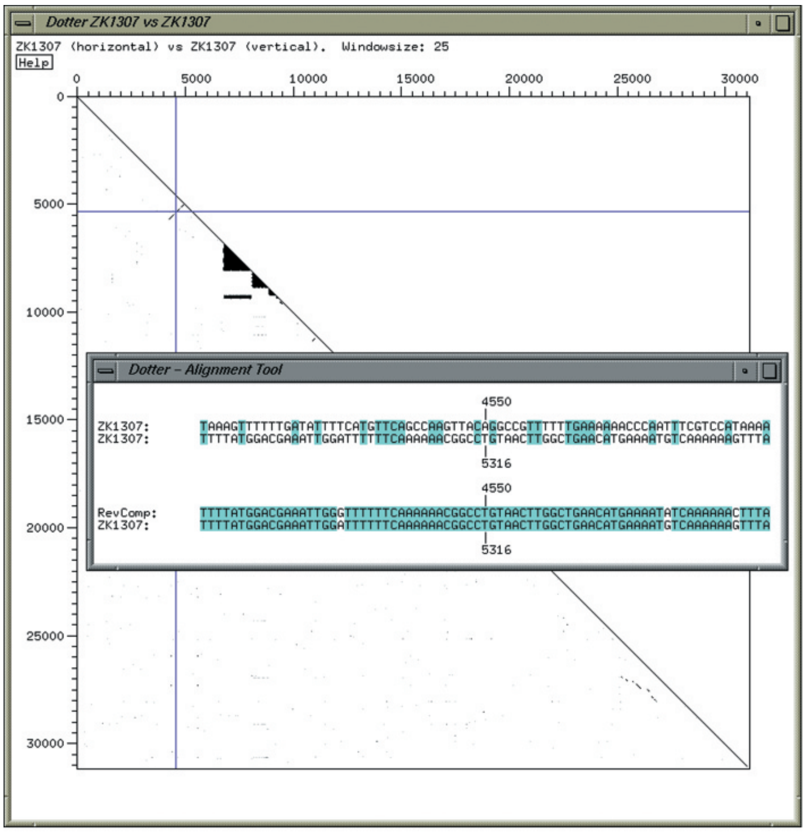
[5] http://www.sanger.ac.uk/dotter.html
[6] ftp://ftp.sanger.ac.uk/pub/acedb

```
integers    N,                           // Length of horizontal sequence
            M,                           // Length of vertical sequence
            α                            // Size of alphabet
            W                            // Span of sliding window

vectors     scoreVec[1..α+1][1..N],      // The score vectors
            newsum[1..N],                // Window-sum vector 1
            oldsum[1..N],                // Window-sum vector 2
            zeroVec[1..N],               // Vector of zeros
            symbVec[1..M]                // Symbols in vertical sequence

pointers    addVec,                      // Pointer to scoreVec to be added
            delVec,                      // Pointer to scoreVec to be subtracted
            tmp                          // Temporary pointer


for i ← 1 to N do
{
  tmp ← oldsum
  oldsum ← newsum
  newsum ← tmp

  addVec ← scoreVec[symbVec[i]]
  if i > W then delVec ← scoreVec[symbVec[i-W]]
  else delVec ← zeroVec

  newsum[1] ← addVec[1]
  for j ← 2 to W do
    newsum[j] ← oldsum[j-1]+addVec[j]
  for j ← W+1 to M do
  {
    newsum[j] ← oldsum[j-1]+addVec[j]-delVec[j-W]
  if newsum[j] > 0 and i > W then
    dot-matrix[i-W/2][j-W/2] ← newsum[j]/W
  }
}
```

It is worth noting that the main operations are all vector additions and subtractions, which would make the program M times faster on an architecture allowing simultaneous vector operations. The above algorithm gives a performance of 5.7 million dots per second on a DEC Alpha AXP 3000/700. This means that a cosmid sequence of 40.000 basepairs can be compared against itself in about 4.5 minutes. Other programs have reported speeds of 0.0005 [18], 0.1 [4], 0.005 [7] and 0.08 [17] million dots/second, albeit on slower hardware. The only program we could benchmark on the same hardware as DOTTER was DIAGON [8] which runs at 0.46 million dots/second.

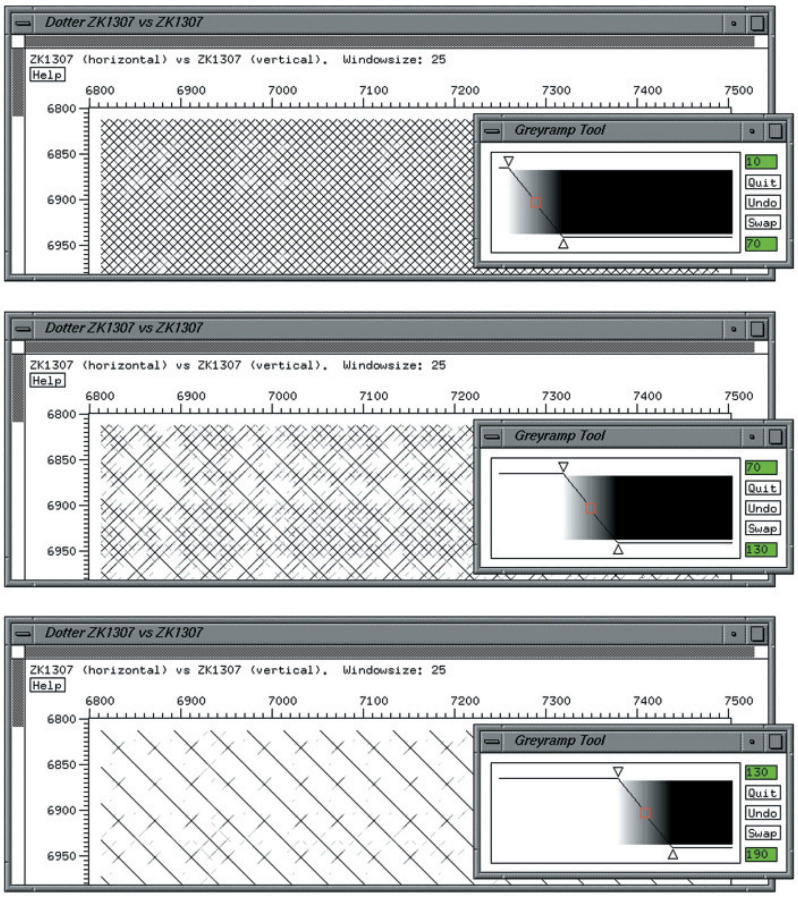
The total memory usage of DOTTER is $(\alpha + 4)$N + 2M plus the dot-matrix itself (1 byte/dot). The memory usage of the dot-matrix is not O(NM) since if NM is large, we only keep a compressed matrix. DOTTER calculates the compression factor based on a user-settable option S, the maximum memory usage of the dot-matrix (default 0.5 Mb). If the product NM is greater than S, we let each pixel represent aa T $\times$ T region of the full matrix, where T is the smallest integer that satisfies $\mathrm{NM}/\mathrm{T}^2 < \mathrm{S}$. Although all the values in the full matrix are calculated, only the maximum value in each T $\times$ T square is kept [18]. This process increases the background noise, but this is readily compensated for by raising the thresholds in the Greyramp tool (see below). If either N or M is greater than the number of horizontal or vertical pixels of the screen, scroll bars will appear to let the user pan through the dot-matrix.

By default, DOTTER sets the window-span to the length of the expected Maximal Segment Pair (MSP), which is calculated for the given sequences and score matrix the following way. Karlin and Altschul showed that for two sequences of length n and m, the MSP score $M(nm)$ has a distribution approximated by $P(M(nm) - (\ln nm)/\lambda > x) \approx 1 - \exp\{-Ke^{-\lambda x}\}$, and provided a method for solving K and λ [19]. The mode of this


Dotter ZK1307 vs ZK1307

```
ZK1307 (horizontal) vs ZK1307 (vertical).  Windowsize: 25
```

```
                                      4550
                                        |
ZK1307:     TAAAGTTTTTTGATATTTTCATGTTCAGCCAAGTTACAGGCCGTTTTTTGAAAAAACCCAATTTCGTCCATAAAA
ZK1307:     TTTTATGGACGAAATTGGATTTTTTCAAAAAACGGCCTGTAACTTGGCTGAACATGAAAATGTCAAAAAAGTTTA
                                        |
                                      5316

                                      4550
                                        |
RevComp:    TTTTATGGACGAAATTGGGTTTTTTCAAAAAACGGCCTGTAACTTGGCTGAACATGAAAATATCAAAAAACTTTA
ZK1307:     TTTTATGGACGAAATTGGATTTTTTCAAAAAACGGCCTGTAACTTGGCTGAACATGAAAATGTCAAAAAAGTTTA
                                        |
                                      5316
```


Dotter ZK1307 vs ZK1307

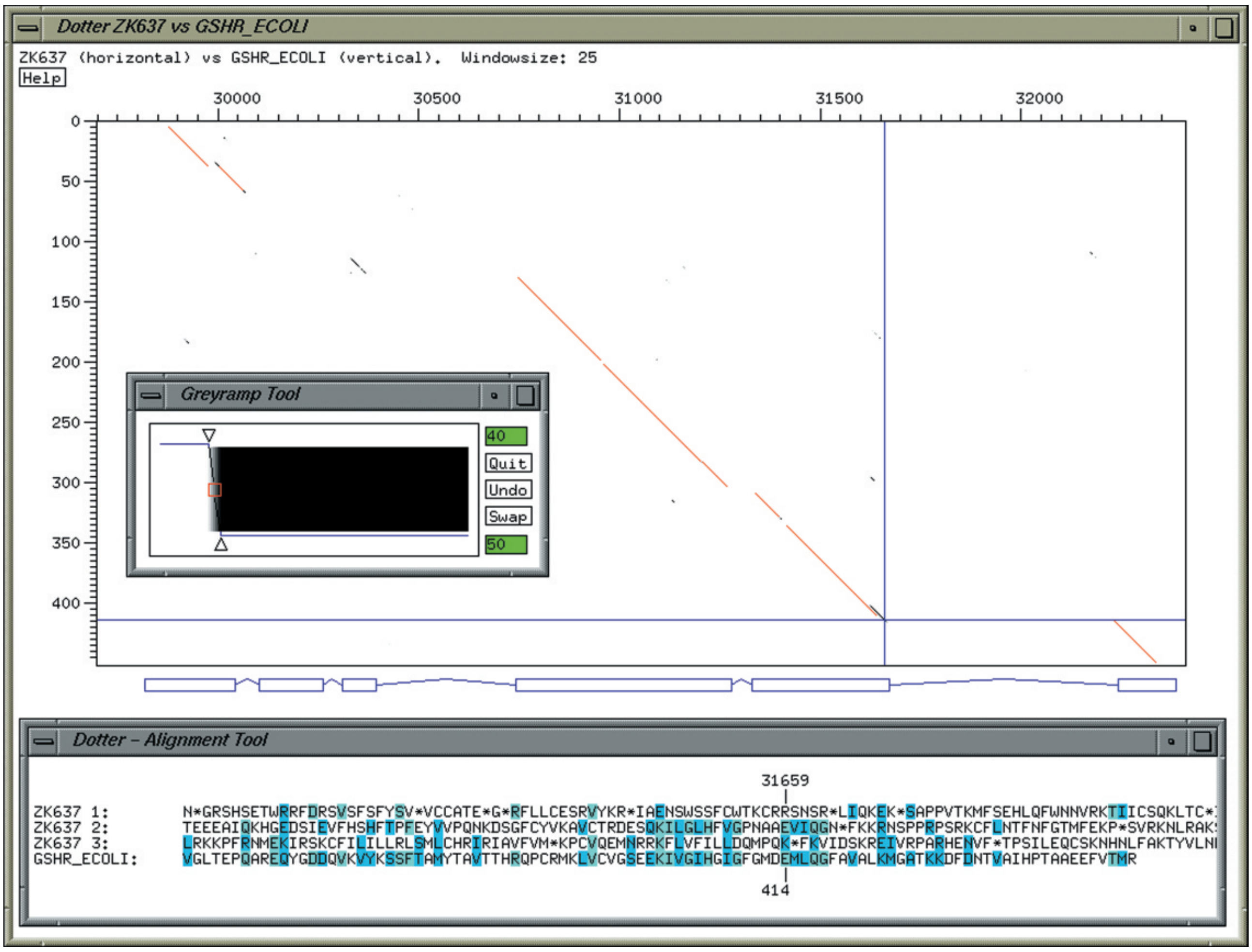
```
ZK1307 (horizontal) vs ZK1307 (vertical).  Windowsize: 25
```

Dotter ZK1307 vs ZK1307

```
ZK1307 (horizontal) vs ZK1307 (vertical).  Windowsize: 25
```

Dotter ZK1307 vs ZK1307

```
ZK1307 (horizontal) vs ZK1307 (vertical).  Windowsize: 25
```



```
ZK637 (horizontal) vs GSHR_ECOLI (vertical).  Windowsize: 25
```

```
                                                                31659
                                                                  |
ZK637 1:     N*GRSHSETWRRFDRSVSFSFYSV*VCCATE*G*RFLLCESRVYKR*IAENSWSSFCWTKCRRSNSR*LIQKEK*SAPPVTKMFSEHLQFWNNVRKTIICSQKLTC*
ZK637 2:     TEEEAIQKHGEDSIEVFHSHFTPFEYVVPQNKDSGFCYVKAVCTRDESQKILGLHFVGPNAAEVIQGN*FKKRNSPPRPSRKCFLNTFNFGTMFEKP*SVRKNLRAK
ZK637 3:     LRKKPFRNMEKIRSKCFILILLRLSMLCHRIRIAVFVM*KPCVQEMNRRKFLVFILLDQMPQK*FKVIDSKREIVRPARHENVF*TPSILEQCSKNHNLFAKTYVLN
GSHR_ECOLI:  VGLTEPQAREQYGDDQVKVYKSSFTAMYTAVTTHRQPCRMKLVCVGSEEKIVGIHGIGFGMDEMLQGFAVALKMGATKKDFDNTVAIHPTAAEEFVTMR
                                                                  |
                                                                 414
```

Fig. 2. DOTTER plot of DNA vs. protein with gene predictions from ACEDB. Shown here is a stretch of genomic DNA from the *C. elegans* cosmid ZK637 (EMBL **Z11115**) compared to the protein glutathione reductase from *Escherichia coli* (SWISSPROT **P06715**). The gene prediction (ZK637.10) was made in ACEDB, but some exons have only very weak homology. Matches found by BLAST/MSPcrunch [20] are superimposed in the dot-plot as red lines. The match at exon 3 was too weak to be reported by BLAST/MSPcrunch, but it is visible in the dot-plot. Also, the BLAST match at the end of exon 5 was extended past an insertion, whereas the dot-plot shows the correct diagonal. The Alignment tool shows the alignment of the three translated forward frames of ZK637 with GSHR_ECOLI at the end of exon 5 (see crosshair position). Frame 2 contains the match missed by BLAST. The calculation took 0.6 seconds.

distribution, or the *expected MSP score* is then $(\ln nm + \ln K)/\lambda$. The expected *score per residue* in an MSP is $R = \Sigma q_{ij} S_{ij}$; $q_{ij} = p_i p_j \exp\{\lambda S_{ij}\}$, where p_i and p_j are the symbol frequencies in the sequences. By dividing the expected MSP score with the expected score per residue we obtain a simple approximation to the expected MSP

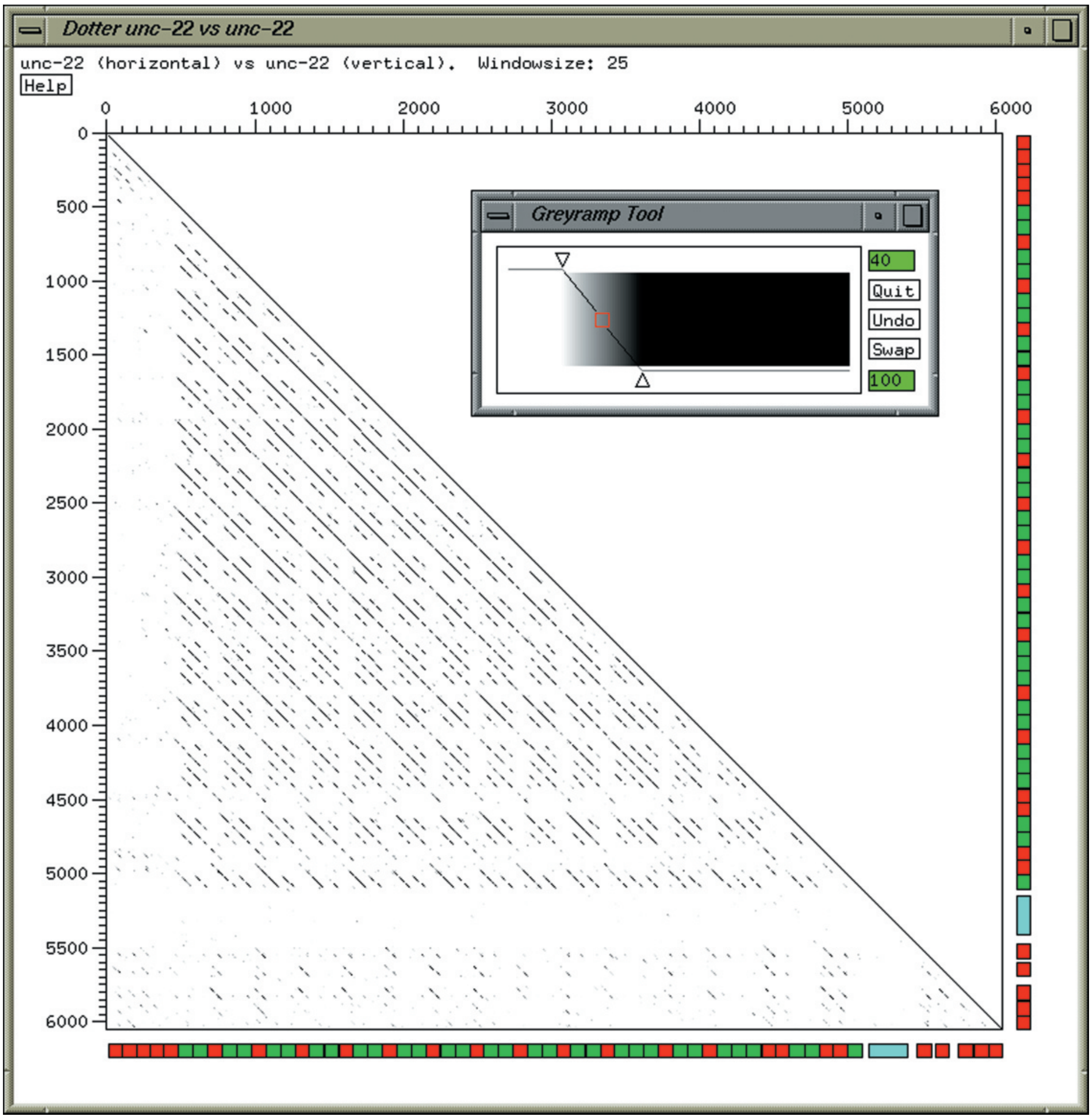
Fig. 1. Dot-matrix analysis of the *C. elegans* cosmid (ZK1307, EMBL **Z47358**) with DOTTER. (a) The entire cosmid compared to itself, with the forward and reverse direction diagonals superimposed. Only half the dot-matrix is drawn since the other half is an identical mirror image. Features that can be seen at this level are an inverted repeat at 4000–6000, a region containing a multitude of small tandem direct and inverted repeats at 6700–9500 and a duplicated gene repeat at 25000–28000. The alignment in both directions at the position of the crosshair is shown in the Alignment tool window in the middle. (b) Zoomed in detail of (a) in a tandem repeat region of about 100 10 bp repeats between 6700 and 8100. The Greyramp Tool is used to view the dot-matrix at different stringencies. The pixel values are 50 times the average residue-score in the window, meaning that a 100% identical match would score 250, given the scoring scheme of +5 for matches and −4 for mismatches. Any dot scoring below the *min* threshold of 10 will be invisible, dots above the *max* threshold of 70 will be completely black, and dots in between will be drawn in a greyscale proportional to their score. (c) If the rendering thresholds are moved up to 70–130, it becomes clear that every 4 of the 10 bp repeats have stronger similarity with each other, suggesting a super-structure repeat unit of 40 bp. (d) Moving the thresholds up to 130–190 shows only the 40 bp repeat structure in the forward direction and only faint inverted diagonals, also with a pitch of 40 bp. The calculation of (a) took 170 seconds and of (b), (c) and (d), which are different renderings of one dot-matrix, 0.1 seconds.

length:

$$\frac{\dfrac{\ln nm + \ln K}{\lambda}}{\sum p_i p_j e^{\lambda \cdot S_{ij}} S_{ij}}$$

For typical sequences and score matrices such as BLOSUM62 for protein and {match = +5; mismatch = −4} for DNA, this usually gives a window-span of about 25 residues. If the above method gives an undesired window-span or fails because λ is undefined for the chosen scoring scheme, it can also be set manually. Because we are interested in local similarities we set n and m to a constant value of 100. This makes the noise density independent of the sequence lengths.

DOTTER can also run in batch mode. In both interactive and batch mode, the dot-matrix and all used parameters can be saved to file and be inspected later. The ability to load dot-matrices from file also makes it possible to



```
unc-22 (horizontal) vs unc-22 (vertical).  Windowsize: 25
```

Fig. 3. Analysis of a highly repetitive protein with symbolic domain annotation. The protein UNC-22, or twitchin (PIR **S07571**) from *C. elegans* is compared to itself. Pixel values are 50 times the average residue-score in the window. The colours of the segments are: green = fibronectin type III domain (FN3); red = immunoglobulin domain (IG); Blue = kinase domain. It is clear that the all the FN3 domains are much more closely related to each other than the IG domains, and that the FN3-FN3-IG cassettes between 1000 and 3000 are more closely related than the other ones. The calculation took 3 seconds.

generate dot-matrices with other programs and read them in DOTTER for interactive inspection. See the World Wide Web address below for details of the format.

2.2. Visualizing the dot-matrix with the Greyramp tool

The Greyramp tool was designed to enhance the visualisation of greyscale images, particularly images with a delicately balanced mix of noise and signal. The simplest form of displaying the score of a dot as a greyscale is to let the intensity be directly proportional to the value of the dot. The Greyramp tool provides two additional features: A min cutoff score, below which all dots get minimum intensity, and a max cutoff score, above which all dots get maximum intensity. For dots scoring between min and max, the dot intensity is linearly proportional to the score. The score shown in the Greyramp tool is the score per residue, i.e. the total score of the sliding window divided by the window-span, multiplied by a scaling factor to use the pixel intensity range 0–255 optimally. By setting this scale factor to $256/5R$, where R is the expected score per residue in an MSP (see above), we place the expected noise level at a fifth (51.2) of the intensity range and thereby make the significance of the pixel intensities roughly the same for different scoring schemes. By starting the Greyramp tool with min = 40 and max = 100, the top of the noise will be just visible, and all scores above twice the expected significant level will be at maximum intensity. Empirically this gives reasonable starting points.

The min and max cutoffs can be changed dynamically and can be controlled independently by point-and-drag actions with the mouse on the little triangles seen in Fig. 1. By dragging the little box in the middle between min and max they are modified simultaneously while keeping the difference between them constant. Minimum intensity is usually white and maximum black, but this can be reversed by the ‘swap’ function. For any setting of the min and max thresholds, the rendered dot-matrix can be printed out on a postscript printer.

DOTTER changes the greyscales on the screen by modifying the colormap cells of 8-bit X-windows displays, which are the most common. Since the colormap cells are not needed on 24-bit graphics, DOTTER will not work on such displays. To work properly, DOTTER needs to allocate 128 of 256 colormap cells for which it can change the displayed colour without other programs becoming discoloured. Situations may arise however, when DOTTER cannot run due to other applications that allocate too many colormaps, in which case they have to be terminated before DOTTER can work. Simultaneous DOTTER jobs on the same display share the same colormaps.

2.3. The crosshair and Alignment tool

A crosshair can be moved either with the mouse or cursor keys around the dot-plot. The extent of a diagonal can be found either by reading the coordinates next to the crosshair or from the rulers on the axes. The sequence alignment of a diagonal can be displayed by moving the crosshair onto it and launching the Alignment tool from DOTTER’s main menu (right mouse button). The Alignment tool displays a residue by residue alignment of the two sequences corresponding to the diagonal around the crosshair. Identical matches are highlighted in bright blue and conservative substitutions in dark blue. If both sequences are DNA, two alignments are possible: of the original sequences and of the reverse complement of the horizontal sequence to the vertical sequence. The two alignments can be shown simultaneously as in Fig. 1. If the horizontal sequence is DNA and the vertical is protein, the three forward frames are translated and superimposed in the dot-matrix, keeping the maximum value in each pixel as described above for compressed matrices. The only way of telling which frame caused a diagonal is to use the Alignment tool, which displays all three reading frames aligned to the protein sequence (Fig. 2).

If the nature of some segments in one or both of the sequences is already known, DOTTER can enhance the analysis by displaying such segments as coloured boxes along the border of the dot-plot, as in Fig. 3. The coloured segments seen in the border are read in from a simple data file with one line per segment. The format is: sequence (1 = horizontal, 2 = vertical), start, end, colour, annotation (**more details**[7]). In combination with the crosshair, the coloured boxes are easy to relate to a particular diagonal.

2.4. Zooming in

It is possible to zoom in to any region in a compressed dot-matrix by dragging with the middle mouse button to delimit a rectangle, or with exact coordinates via a dialogue window. A new DOTTER job will then be spawned for the selected region only. The parent DOTTER job will not be superseded but will remain intact on the screen. The two dot-plots will be independent of each other so that either can be killed without affecting the other one. Since all simultaneous DOTTER jobs share the same colormaps, any Greyramp tool will control the greyscale rendering of all active dot-plots.

2.5. Displaying high-scoring segments

Calculating the full dot-matrix, as described above, has two drawbacks: it is slow for very long sequences, and it does not display the maximum high-scoring extent of the diagonals. Sometimes it is informative to try to extend a diagonal in both directions until the total score doesn’t increase further. The ungapped alignment giving the maxi-

[7] http://www.sanger.ac.uk/dotter.html

mum score is called a high-scoring segment pair (HSP). The BLAST programs [15] search for HSPs in a fast, heuristic fashion. Instead of replicating the BLAST algorithm, DOTTER simply reads in HSPs reported by BLAST and draws them in the dot-plot as in Fig. 2, similarly to PLFASTA [12] for FASTA output. Here it is accomplished via the BLAST output viewer BLIXEM [20], which constructs a multiple alignment of HSPs reported by BLAST and displays it graphically in a scrollable window. The advantage of this is that BLIXEM first can give an overview of all sequences that match a given query. The most interesting homologies can then be explored in much finer detail by calling up DOTTER 'on the fly'. BLIXEM hands the HSPs over to DOTTER, which can display the HSPs in two different ways: by greyscale according the total HSP score, or by monochrome red lines which can be superimposed over the full dot-matrix. It is also possible to superimpose four different shades of red to reflect the score of the HSP.

2.6. Using DOTTER for gene prediction

The genomic database **ACEDB** [8] [21] allows interactive gene modelling, with full display of relevant features such as splice sites, open reading frames, segments of high coding potential, sequence homology, etc. If the gene in question has homologous sequences, the multiple alignment of the homologues can be viewed by calling up BLIXEM from ACEDB, which also passes on the tentative gene prediction coordinates. For a more detailed analysis of how the homology fits with the gene prediction, the coordinates of the predicted gene are also passed on from BLIXEM to DOTTER, which then displays the dot-plot comparison between the genomic DNA where the gene was predicted and the homologous protein (Fig. 2). Having the gene prediction displayed in the dot-plot significantly aids the ability to accept weakly conserved exons, and to reject ones that are inconsistent with the homology.

3. Application

Sequenced cosmids from the *C. elegans* genome sequencing project [22] are routinely compared to themselves with **DOTTER** [9] for analysis of the extent and nature of direct and inverted DNA repeats. Such repeats are interspersed throughout the genome, and there are many different recurring families [23]. For example, the *C. elegans* cosmid ZK1307 (Fig. 1) contains several repeat families: $33+4$ copies of a 40-mer, 22 copies of a 35-mer, 21 copies of a 15-mer and 2 copies of a 123-mer which contain 5 copies of an 11-mer in the middle. Naclerio et al. previously described the first 3 of these repeat families and named them RcC9, Rc35 and RcD1, respectively. The 40 bp repeat RcC9 [24], between 6750 and 8050, shown in detail in Fig. 1b–d is especially interesting since it has a less strongly conserved subunit of 10 bp which itself is palindromic, giving a minimal repeat unit in alternate orientations of only 5 bp: -TTC-. The smaller repeat units are however much less conserved than the 40 bp repeat. At very low stringency the dot-plot hence shows 10 bp spaced diagonals in both orientations (Fig. 1b). As the stringency is raised (Fig. 1c–d), the 10 bp spaced diagonals fade away, leaving only the strongest conserved 40 bp repeats in the dot-plot.

For arrays of tandem repeats such as this, DOTTER makes it very easy to find the start and end of the repetitive unit and the number of repeats, which is especially important for constructing high-quality multiple alignments. As illustrated in Fig. 1, it is often far from trivial to determine the length of the main repeat unit, since multiples or fractions thereof are plausible units too. With the Greyramp and Alignment tools, this becomes a relatively easy task.

The need for a dot-plot program that can compare DNA to protein sequences was also prompted by the *C. elegans* genome project, where most primary protein homology analysis is carried out by comparing DNA to protein. The reason for doing this is that using predicted coding segments may miss homologies if the gene prediction was incorrect. Database searching is usually done with the program BLASTX in conjunction with the filtering program MSPcrunch [20] to increase sensitivity and selectivity. The DNA-protein HSPs are then aligned in the X-windows viewer BLIXEM. Integration of DOTTER into ACEDB and BLIXEM hence made it natural to carry over the DNA vs. protein philosophy to DOTTER, as shown in Fig. 2. One could envisage using a different colour for each translated frame, but given that real homologies are normally confined to a single frame, and that the frame can easily be determined with the Alignment tool, we found the best solution was to leave them in the standard greyscale colours. The exons and introns of the gene prediction are shown just below the dot-plot border.

Fig. 3 shows a self-comparison of the protein UNC-22 or twitchin, a large muscle protein which probably interacts with myosin [25] [26]. It consists of repeated fibronectin type III (FN3) and immunoglobulin (IG) domains and one kinase domain. At the N- and C-termini, five tandemly repeated IG domains are present, while the interior contains repeated 'cassettes' of usually two FN3 and one IG domain. With the coloured segment boxes, it is easy to see how the similarity levels vary for the different domains. For instance, while the FN3 and IG domains in the N-terminal portion of the cassette repeat region are very similar, they are less conserved towards the ends. Especially the IG domains are very poorly conserved except in the middle of the cassette region. The five

[8] ftp://ftp.sanger.ac.uk/pub/acedb
[9] http://www.sanger.ac.uk/dotter.html

N-terminal IG domains are more similar to each other than to other ones, whereas for the five C-terminal IG domains this is not the case. The dot-plots in Figs. 2 and 3 were generated using the score matrix BLOSUM62 [27].

4. Discussion

DOTTER[10] is a new type of dot-plot program which is well suited to handle demanding homology analysis tasks involving weak and difficult to assess matches in both traditional protein or DNA comparisons and in more complex situations when genomic DNA is compared to proteins or DNA. Its main strength is that the dot-matrix only has to be calculated once, after which the stringency thresholds are varied dynamically, avoiding tedious reiteration of the dot-matrix calculation. This is particularly useful when no optimal stringency exists, for instance if a diagonal can only be seen when the background noise is also visible. Such diagonals may still be biologically significant if they make good sense with other diagonals and/or if they contain important key residues. In cases like this, it is desirable to view the dot-plot under many different stringency conditions and be able to change them in a scrolling fashion.

The program XSauci [28] also uses colormaps for dynamic threshold control, for a variant of dot-plots called ‘correlation images’, which transforms diagonals to horizontal lines. XSauci uses greyscales differently than DOTTER however, in that the pixel intensity reflects the length of a match instead of the score, and it employs only one threshold.

The integration of DOTTER into the multiple alignment viewer for BLAST matches, BLIXEM, makes a very powerful combination. With the add-on MSPcrunch, BLAST usually picks up at least one local match to homologous sequences, but may miss weak matches or matches to repeated domains. DOTTER can then be called up directly from BLIXEM for a particular protein to show the true extent of the homology. This system provides very efficient and comfortable sequence homology analysis, with a minimal risk of overlooking similarities or assessing them incorrectly.

Alignment algorithms based on dynamic programming are a popular method of pairwise sequence similarity analysis which can be very sensitive if the gap weights are set correctly. However, for weak similarities the alignment is often very vulnerable to small changes in the gap weights, and often only a narrow range of parameters gives the correct alignment [29] [30]. Dot-plots do not suffer from this problem, since no attempt is made to string matching segments together with gaps in between. Several users have asked if it would be possible to generate a gapped alignment by dynamic programming from DOTTER. Since this would not improve over the standard implementations of dynamic programming, we have not included this feature. One might envisage however, that the user could select a number of diagonals, which are considered relevant. These segments could then be strung together in an alignment, possibly using dynamic programming to fill in the gaps, but allowing interactive control of the alignment path [31].

DOTTER is available by **anonymous FTP**[11], **World Wide Web**[12] or by sending E-mail to `esr@sanger.ac.uk`. ACEDB is available by **anonymous FTP**[13].

[10] http://www.sanger.ac.uk/dotter.html

[11] ftp://ftp.sanger.ac.uk/pub/dotter

[12] http://www.sanger.ac.uk/dotter.html

[13] ftp://ftp.sanger.ac.uk/pub/acedb

Acknowledgements

We are grateful to Friedemann Wobus for work on the Greyramp tool and various other image handling software in the ACEDB graphics library, and to the *C. elegans* sequencing consortium, in particular Mary Berks for sequencing the cosmid ZK1307. We thank Gos Micklem and Sean Eddy for helpful discussions and John Sulston for reading the manuscript. The Sanger Centre is supported by the Wellcome Trust and the MRC.

References

[1] Fitch, W.M. (1969) Locating gaps in amino acid sequences to optimize the homology between two proteins. *Biochem. Genet*. **3**, 99–108.

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

[3] McLachlan, A.D. (1971) Test for comparing related amino acid sequences. Cytochrome c and cytochrome c551. *J. Mol. Biol*. **61**, 409–424.

[4] McLachlan, A.D. (1983) Analysis of gene duplication repeats in the myosin rod. *J. Mol. Biol*. **169**, 15–30.

[5] McLachlan, A.D. and Boswell, D.R. (1985) Confidence limits for homology in protein or gene sequences. The c-myc oncogene and Adenovirus E1A protein. *J. Mol. Biol*. **185**, 39–49.

[6] 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. *Comput. Appl. Biosci*. **3**, 25–30.

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

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

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

[10] Reisner, A.H. and Bucholtz, C.A. (1988) The use of various properties of amino acids in color and monochrome dot-matrix analyses for protein homologies. *Comput. Appl. Biosci*. **4**, 395–402.

[11] Zuker, M. (1991) Suboptimal sequence alignment in molecular biology. Alignment with error analysis. *J. Mol. Biol*. **221**, 403–420.

[12] Pearson, W.R. and Lipman, D.J. (1988) Improved tools for biological sequence comparison. *Proc. Natl. Acad. Sci. USA* **85**, 2444–2448.

[13] Schwartz, S., Miller, W., Yang, C.M. and Hardison, R.C. (1991) Software tools for analyzing pairwise alignments of long sequences. *Nucleic Acids Res*. **19**, 4663–4667.

[14] 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.

[15] Altschul, S.F., Gish, W., Miller, W., Myers, E.W. and Lipman, D.J. (1990) Basic local alignment search tool. *J. Mol. Biol*. **215**, 403–410.

[16] Durbin, R. (1995) *(unpublished)*,

[17] Karreman C. (1992) A dotplot program for the Atari ST, for the analysis of DNA and protein sequences. *Comput. Appl. Biosci*. **8**, 75–77.

[18] Pustell, J.M. and Kafatos, F.C. (1982) A high speed, high capacity homology matrix: zooming through SV40 and Polyoma. *Nucleic Acids Res*. **10**, 4765–4782.

[19] Karlin, S. and Altschul, S.F. (1990) Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes. *Proc. Natl. Acad. Sci. USA* **87**, 2264–2268.

[20] Sonnhammer, E.L.L. and Durbin, R. (1994) A workbench for large-scale sequence homology analysis. *Comput. Appl. Biosci*. **10**, 301–307.

[21] Durbin, R. and Thierry-Mieg, J. (1995) *(unpublished)*,

[22] Wilson, R., Ainscough, R., Anderson, K., Baynes, C., Berks, M., Bonfield, J., Burton, J., Connell, M., Copsey, T., Cooper, J., Coulson, A., Craxton, M., Dear, S., Du, Z., Durbin, R., Favello, A., Fulton, L., Gardner, A., Green, P., Hawkins, T., Hillier, L., Jier, M., Johnston, L., Jones, M., Kershaw, J., Kirsten, J., Laisster, N., Latreille, P., Lightning, J., Lloyd, C., Mortimore, B., O'Callaghan, M., Parsons, J., Percy, C., Rifken, L., Roopra, A., Saunders, D., Shownkeen, R., Sims, M., Smaldon, N., Smith, A., Smith, M., Sonnhammer, E., Staden, R., Sulston, J., Thierry-Mieg, J., Thomas, K., Vaudin, M., Vaughan, K., Waterston, R., Watson, A., Weinstock, L., Wilkinson-Sproat, J. and Wohldman, P. (1994) 2.2 Mb of contiguous nucleotide sequence from chromosome III of C. elegans. *Nature* **368**, 32–38.

[23] Naclerio, G., Cangiano, G., Coulson, A., Levitt, A., Ruvolo, V. and La Volpe, A. (1992) Molecular and Genomic Organization of Clusters of Repetitive DNA Sequences in Caenorhabditis elegans. *J. Mol. Biol*. **226**, 159–168.

[24] La Volpe, A., Ciaramella, M. and Bazzicalupo, P. (1988) Structure, evolution and properties of a novel repetitive DNA family in Caenorhabditis elegans. *Nucleic Acids Res*. **16**, 8213–8231.

[25] Benian, G.M., Kiff, J.E., Neckelmann, N., Moerman, D.G. and Waterston, R.H. (1989) Sequence of an unusually large protein implicated in regulation of myosin activity in C. elegans. *Nature* **342**, 45–50.

[26] Benian, G.M., L'Hernault, S.W. and Morris, M.E. (1993) Additional Sequence Complexity in the Muscle Gene, unc-22, and Its Encoded Protein, Twitchin, of Caenorhabditis elegans. *Genetics* **134**, 1097–1104.

[27] Henikoff, S. and Henikoff, J.G. (1992) Amino acid substitution matrices from protein blocks. *Proc. Natl. Acad. Sci. USA* **89**, 10915–10919.

[28] Nedde, D.N. and Ward, M.O. (1993) Visualizing relationships between nucleic acid sequences using correlation images. *Comput. Appl. Biosci*. **9**, 331–335.

[29] Argos, P. and Vingron M. (1990) Sensitivity comparison of protein amino acid sequences. *Methods Enzymol*. **183**, 352–365.

[30] Vingron, M. and Waterman, M.S. (1994) Sequence alignment and penalty choice. Reviews of concepts, case studies and implications. *J. Mol. Biol*. **235**, 1–12.

[31] Rechid, R., Vingron, M. and Argos P. (1989) A new interactive protein sequence alignment program and comparison of its results with widely used algorithms. *Comput. Appl. Biosci*. **5**, 107–113.