Volume 12 Number 1 1984 Nucleic Acids Research

The diagonal-traverse homology search algorithm for locating similarities between two sequences

C.Thomas White, Stephen C.Hardies, Clyde A.Hutchison III and Marshall H.Edgell

Department of Microbiology and Immunology, Curriculum in Genetics, Program in Molecular Biology and Biotechnology, University of North Carolina, Chapel Hill, NC 27514, USA

Received 12 August 1983

ABSTRACT

We present a fast computer algorithm for finding homology between two DNA sequences. It generates a two-dimensional display in which a diagonal string of dots represents a stretch of homology between the two sequences. Our algorithm performs the search very rapidly, and has no internal data storage requirement except for the sequences themselves. These characteristics make it particularly well suited for execution on microcomputers. Without slowing execution, the matching criterion can be that a specified fraction of contiguous bases must be identical. Even with gapped sequences, we have found large search windows to be surprisingly good for detecting poor homologies with nearly complete background suppression. A diagonal search pattern is used that reports the finds in a compact and logically ordered form. A simple and rapid plotting algorithm for unsophisticated printers is also reported.

INTRODUCTION

A frequent problem in molecular biology is to determine whether two different sequences are related. The sequences in question may be experimentally determined amino acid sequences or nucleotide sequences. Two-dimensional plots representing all of the homologous segments between two sequences have appeared in the literature under a variety of names (1-10). We will refer to them as "dotplots". The horizontal and vertical axes of the plot correspond to positions within the respective sequences. Each (x,y) point within the plot is marked with a dot if the corresponding x-th position of one sequence matches the y-th position of the other. Extended homologies between the two sequences will appear as a diagonal string of dots.

The wealth of recently acquired DNA sequence data provides numerous applications for computerized comparison techniques. Dotplots can be used to address questions of evolutionary relationships, rearrangement, and movement of sequence through gene conversion among related genes (3,12-16). Dotplots promote discovery because their visual form interacts well with the scientist's intuition. They are also useful in mounting preliminary surveys over large amounts of sequence because of their algorithmic speed and simplicity.

A computer algorithm to generate dotplots for analyzing the relationships among cytochrome c amino acid sequences was written by Gibbs and McIntyre (9) and also applied to nucleotide sequences. It used the simplest possible criterion for displaying a dot, which is that the two individually compared amino acids or nucleotides be identical. This strategy produces a considerable background of chance matches, particularly for nucleotide data. McLachlan (2) described the suppression of this background by requiring that individual dots correspond to a short "span" of homology to merit display. This strategy of comparing multiple nucleotides or amino acid residues per printed dot is common to subsequent programs (3-10). We will refer to the size of the region compared for a single dot as a "window", and to the number of individual matches required within the window for the dot to be printed as the "stringency". Unfortunately, the use of windows to suppress background has often led to significantly increased execution time.

When sequences diverge, they accumulate small insertions and deletions as well as substitutions. Therefore, alignment of the sequences to reveal their evolutionary relationships must include gap positions; places where one sequence has no homolog to the other. Discovering where to put the gaps can be complicated, and mathematically defined alignment algorithms have been developed for this purpose (17-19). In dotplots, gaps cause the diagonal string of dots to break and shift to a new diagonal. When a gap falls in a search window, it may prevent detection of homology. One finding in this paper is that gaps caused less of a problem with very large search windows than we had expected.

Because we wanted to use our dotplotting system on a microcomputer, we had to design the software to compensate for the limitations of the hardware. The programs had to be very fast, but still effectively suppress noise. It had to not require large amounts of memory devoted to storing arrays. We also wanted it to produce plots without requiring sophisticated graphics equipment. To meet these ends, we devised a search algorithm, most similar to that used by Staden (1Ø), which allowed a simple unweighted window and variable stringency without any penalty in execution time. We rearranged the typical left to right search pattern to one that traverses the full length of each diagonal in order. This allows direct reporting of the results in a tabular form where the exact coordinates and extents of homologies are obvious. We also designed a simple and rapid plotting algorithm that rotates the plot by 45 degrees, thereby producing good quality scaled plots with standard printing equipment.

METHODS

Internal Data Representation.

If s1 and s2 are each members of a sequence, the search algorithm uses a boolean function MATCH(s1,s2) that evaluates to <true> if the elements are considered to match. Otherwise, MATCH(s1,s2) evaluates to <false>. In order to obtain a MATCH function that executes rapidly and correctly matches ambiguous positions, we first convert the internal representation of the DNA sequence into a bit string. That is, 4 consecutive bits represent AGCT such that 1ØØØ is A, Ø1ØØ is G, 11ØØ is A or G, etc. For a fast MATCH function, we then use the elementary machine instruction that performs a logical AND.

Definition of the Intermediate Data Structure.

The results of the search program are defined both by what is found and the order in which the finds are reported.

Preliminary Definitions. Consider sequence A to be indexed by the integer x, such that 1 <= x <= length(A), and A[x] refers to the x-th element of sequence A. Consider sequence B to be similarly indexed by the integer y.

Definition: Part A. For the comparison of sequences A and B consider a homology criterion to be given requiring M matches within a window of W consecutive bases. (x,y) defines the position of a matched window if and only if:

(1) x is in the range 1 to length(A)-W+1, and

(2) y is in the range 1 to length(B)-W+1, and

(3) for i=0 to W-1,
MATCH(A[x+i],B[y+i]) is true at least M times.

Then the search program is confined to reporting all sets of positive integers (X,Y), and only those sets, which meet the following criteria.

(4) (X,Y) must be a matched window.

(5) There must not be a matched window with coordinates (X-1,Y-1).

Each successful report is in the form (X,Y,L,N) where:

(6) L is the maximum length such that each window (X+i,Y+i) in the range i=0 to L-W is a matched window.

(7) N is the total number of times that for i=0 to L-1, MATCH(A[X+i],B[Y+i]) is true.

Definition: Part B. The reported sets are ordered such that for any two of them, (X1,Y1,L1,N1) precedes (X2,Y2,L2,N2) if and only if:

either (X1-Y1 > X2-Y2),
or (X1-Y1 = X2-Y2, and X1 < X2)

Part A of the above definition requires that all segments continuously meeting the M out of W match criteria are to be reported, so long as they are not themselves contained within a larger matching segment. The specification in Part B is given with the expectation that the order can be taken advantage of by modules processing this structure. Lines 2 and 3 above would exclude finding homologies that cross the arbitrary end point of circular sequences. To avoid that we first extend circular sequences by a circular permutation equal to W-1.

RESULTS AND DISCUSSION

Advantages of the Diagonal-Traverse Homology Search Algorithm.

Increase in Speed. The driving motivation behind the development of this algorithm was a desire to reduce the time required to perform homology searches and produce dotplot maps. This motivation was instilled in us by the desire to use microcomputers that run orders of magnitude slower than a typical mainframe. We achieved a great enough speed that even the smallest and slowest microcomputers can be used to compare sequences whose lengths are in the tens of kilobases. A four megahertz Z-8Ø-based machine in our lab searched two five-kilobase sequences in under twenty-five minutes to produce the largest dotplots in this paper. The smaller ones were produced in a under a minute.

A number of things were done to keep the execution time of the search down. We chose a noise suppression system that uses a simple unweighted window of variable length and a variable stringency, and does not cause an increase in execution time with increased window size. This same strategy was also used in the design of a program described by Staden (1Ø). As a result, the search time is proportional only to the product of the length of the two sequences and is independent of the length of the search window. We reordered the search and simplified the method of reporting the results to reduce the number of operations required in the elemental comparison operation. This part of the algorithm determines the overall execution time because the program spends most of its time repetitively executing it. Our algorithm executes only 3Ø machine instructions as it loops through the elemental compare operation, resulting in an execution time on a 4 megahertz machine of the product of the sequence lengths times $Ø.6 \times 1Ø^{-6}$ min.

The Utility of Large Search Windows. With each search being so economical to perform, it became much easier to compare the results of many searches of the same sequences with different search stringencies. We had previously found with a slower program (12) that a good strategy for discovering homologies between highly diverged sequences is to use low stringency coupled with large search windows. However, one expects large windows to eventually run into problems because of gaps. A rather surprising discovery we have made with our faster algorithm is that windows even larger than 5Ø base pairs work well to distinguish poor homology from background. In fact, background can be nearly eliminated without losing homology between distantly related sequences (Figures 1 and 2.)

SV4Ø to polyoma comparisons have been used as a benchmark

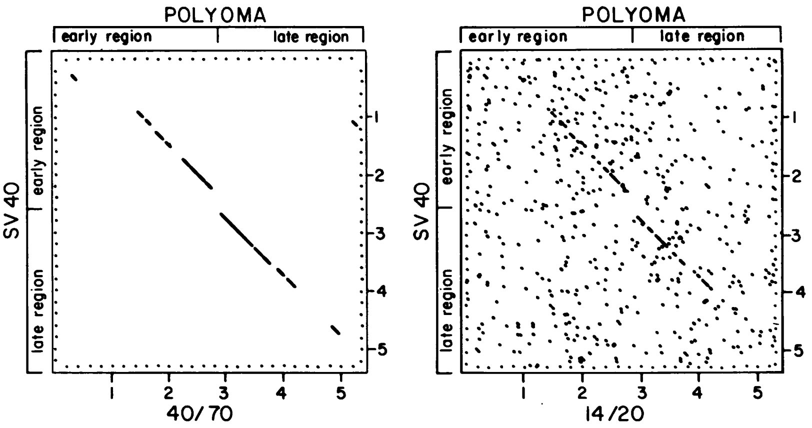

Figure 1. Dotplots of SV40 versus polyoma genomes with large and small windows. The sequences are 5' to 3' from left to right and top to bottom, respectively. Dots around the border mark off increments of 200 base pairs.

for search algorithms with search windows of 5 (3) or 40 (9) in length. Figure 1 shows two dotplots each comparing the nucleic acid sequences of SV40 (20) on the vertical axis versus polyoma

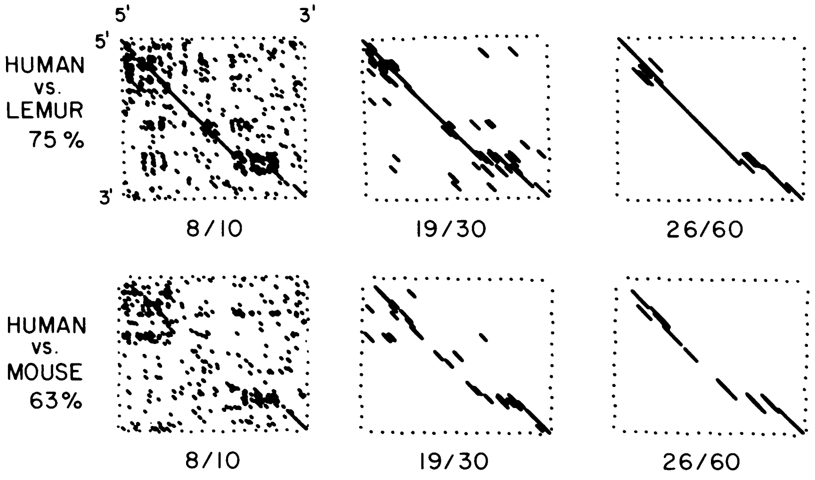

Figure 2. Dotplots of beta-globin IVS 2 sequences with 3 different sized windows. The genes are human delta, lemur pseudo delta, and mouse beta h2. Dots around the border mark off increments of 50 base pairs. The indicated percent homologies were calculated from aligned sequences and without counting gap positions.

virus (21) on the horizontal axis. The polyoma sequence was complemented relative to the published sequence to put it in the same orientation as the SV40 sequence. On the dotplot on the left, each line segment represents a match of at least forty identical bases within a region seventy bases in length. The dotplot on the right differs only in that it displays regions of fourteen matching bases within segments twenty bases in length. Several dotplots were produced using each window size, but with different stringencies (data not shown). The results pictured (Figure 1) were judged to be the best stringencies for each window size. Clearly the larger window size has given less background, and gaps have not yet become a serious problem. Also, although the search time for any window size is the same, the plotting time increases proportionally to the number of finds. In other words, the overall execution time actually goes down for larger windows.

We also asked if large window sizes were beneficial for matching noncoding sequences where the ratio of gaps to base changes is higher (Figure 2). The nucleic acids shown are from the second intervening sequence (IVS 2) of delta-like beta-globin genes of three mammalian species (14,16,22,23). IVS 2 between human and lemur exhibits about 75% homology, while only 63% of the residues are the same for this region between humans and mice. These percent homologies were calculated from the aligned sequences ignoring gap positions. Again the larger search windows are able to detect either level of homology with nearly complete suppression of noise. Again, the best stringency for a given window size was determined empirically.

A number of useful features about dotplots are illustrated by the IVS 2 comparisons (Figure 2). In each case the significance of the major finds can be judged intuitively relative to the random noise elsewhere on the plot. Since all possible alignments are represented somewhere on the plot, the possibility of alternate good alignments can be discerned. For example, the shift of the major string of finds near the bottom right of each of the plots in Figure 2 represents the fact that the lemur and mouse genes each are missing sequences present in human delta at this site. Particularly in the shorter window searches, it can be seen that there are multiple alternative alignments in this region. Inspection of the sequences revealed that the length change could best be described as expansion or contraction of a simple internally repetitive sequence near the 3' end of delta-like IVS 2's (16). The more dense box of background matches in the upper left of each plot is because the sequences are very pyrimidine rich in that region. The statistics for chance matching are altered in regions of unbalanced base composition. The appearance of dense blocks or bands of background matches on the dotplot warns the investigator that the alignment through that region is necessarily of lower confidence. Although we find the cleaner results with large windows useful for many purposes, examination with smaller windows and less than optimal stringencies are often valuable for discovering features of the kind illustrated above.

Description of the Diagonal-Traverse Algorithm.

The Order of Homology Discoveries. A feature of the diagonal-traverse search algorithm is that homologous segments are discovered and ordered by diagonals rather than by horizontal position. This order is exploited both to radically compact the information flow out of this step, and to simplify most kinds of downstream analysis including the plotting. Each diagonal line segment is reported as a position, length, and strength of match, rather than as a string of individually defined dots.

With respect to their arrangement on a dotplot (Figure 3, Top), the search is performed along each 5' to 3' diagonal (directed at -45 degrees), taken in order from the short diagonal in the upper right corner of the plot to the one in the lower left. The segments in Figure 3 are numbered with respect to this order of discovery. This differs from all other programs we have seen in the literature, in that they progress in horizontal rows from left to right, ordered from top to bottom.

The middle portion of Figure 3 represents a snap-shot of the state of the search in progression along the diagonal including segments 8 and 9. All segments found along any such diagonal have in common the same global alignment of the two sequences. The portion of each sequence which includes this region is displayed, demonstrating the alignment in effect for this

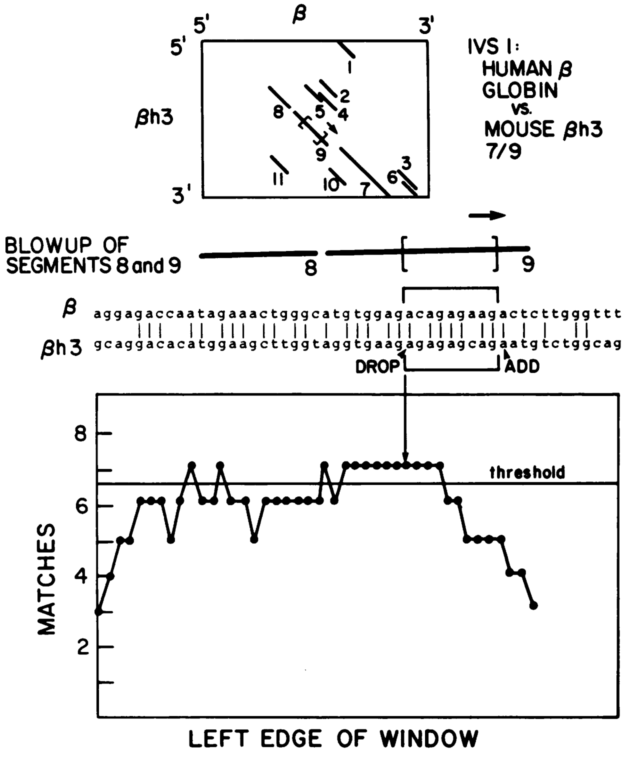

Figure 3. Demonstration of the diagonal-traverse search algorithm. The finds are numbered on the dotplot at the top in the order they were encountered by the search routine. The bracket and arrow show the position of the search window at the instant illustrated below. Segments 8 and 9 are shown reoriented horizontally below the dotplot and aligned with the sequence being searched. During the next search cycle the window will step forward one base on both sequences and recalculate the enclosed homology by dropping one basepair and adding one basepair. Because the homology will still be over the threshold, the extra length will be appended to the output record corresponding to segment 9. The graph at the bottom shows the number of matches found at each position as this particular diagonal was traversed. An output record was generated each time the number of matches climbed over the threshold and then fell back below it. Segments 8 and 9 are both actually composed of 2 overlapping segments. The sequences are IVS 1 from human beta-globin (24) and mouse beta h3 (25).

diagonal.

The search window, at the moment captured by the snap-shot, is denoted by the brackets covering a portion of segment 9. Since seven of the nine positions compared in this window are equivalent, the homology criterion is satisfied at this point. The next window to be evaluated is that which is one to the right of the current one, and therefore requires no realignment of the sequences. Its match total can be computed from the current total by subtracting the contribution of the left-most position and adding the contribution of the first position to the right of the current window. Since the other positions inside the window are not reevaluated, the speed of the search is independent of window size and stringency.

The graph in Figure 3 plots the match total of each window in the neighborhood versus the sequence position corresponding to its left end. Where the total first exceeds the threshold line marked on the graph, the sequence within that window, by definition, satisfies the homology criterion. A matching segment will be reported with the coordinates of this initial matching window and a length accounting for the number of successive adjacent windows that also exceed the threshold. This is the basis for the information compaction previously mentioned. As a further complication, it may be noted that the line segments marked 8 and 9 (Figure 3) are each actually composed of two overlapping segments.

Organization of the Program System.

We considered the job of producing dotplots sufficiently complex to merit breaking the task into smaller pieces. Important advantages are gained in the design of such a programming task by its conceptualization as an interrelated set of smaller tasks. To the extent that these smaller tasks really do different things, they may be developed independently from one another and deserve the attribute of being called mutually "orthogonal". The smaller tasks are easier to write and debug; they are easier to modify and recombine with other tasks for future uses; they are easier to understand and document; and their use of machine resources is easier to define and optimize. Division into orthogonal tasks is considered good programming practice in any system, and it proved to be particularly useful for making maximum use of our microcomputer's resources.

Our dotplotting system includes 3 free-standing programs (Figure 4). The diagonal-traverse search program has the

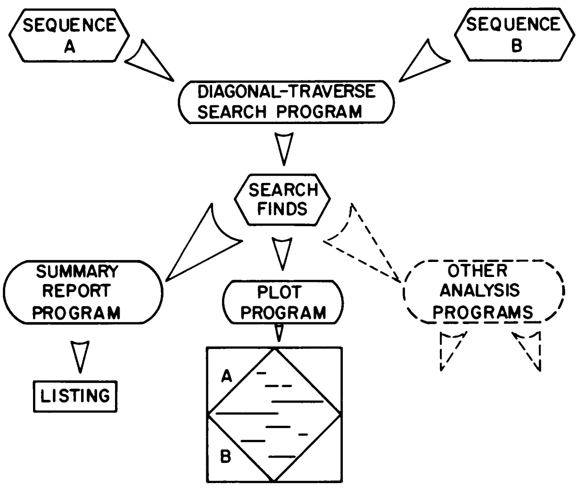

Figure 4. Organization of the diagonal-traverse dotplotting system.

responsibility of loading the sequences to be searched into memory, performing the search, and saving the results on the mass storage device. Alternatively, the intermediate file of search finds may be replaced by a pipe where multi-programming is available. The display and analysis programs are not resident in memory during the search, thus increasing the amount of sequence that can be handled by the microcomputer. This division of labor also means that the printing device need not be present during the most time consuming part of the analysis. This is an important benefit in an environment such as ours where a single printer must service a number of instruments. The intermediate file of search finds can then be utilized for several kinds of subsequent analyses. In our system, a tabular listing can be made for the purpose of extracting the exact coordinates of matches; and, of course, the plotting program itself can be used. The dashed portion of Figure 4 represents other possible uses of the intermediate file that we can envision. It might be subjected to a different kind of noise filter, analyzed by statistical routines, or used to guide an automated alignment routine.

The Intermediate Data Structure.

The data structure labeled "SEARCH FINDS" in Figure 4 is a critical feature of our design. In traditional dotplot searches this construct can become very large. Partly as a result of this, other implementations have avoided producing this structure as a distinct file. Instead they have incorporated both the search and plotting functions into one program. Our intermediate files tend to be short for two reasons. One is that the individual records have a naturally compact structure. The other is that because we are getting such good noise suppression with the larger windows, we seldom make plots containing large numbers of finds.

Another feature of this structure is that matching segments along the same diagonal (those found under the same global alignment) are reported one after the other, and those on nearby diagonals are found just before or afterwards. This order is well suited for most kinds of further analysis, and may be exploited, as explained later, in the production of dotplots. The order of this structure is a reflection of the order in which the homology search is performed, and thus requires no intermediate sorting step.

Analysis of the Intermediate Data Structure.

The intermediate file of search finds can be directly examined to extract the exact coordinates of a particular matching segment for further study. Table I shows the contents of the intermediate file generated during the construction of Figure 3. The coordinates X and Y, the length of the find, and the number of bases matching are reported directly as produced by the search routine. The diagonal number is equal to X-Y and uniquely identifies the diagonal on the plot on which the segment was found. Relative phase is simply the the diagonal number of the previous find minus the diagonal number of the current find. Because of the order of the finds in the file, matches that can be put together into a single alignment will be grouped together. They will appear as a cluster of finds with relative phase equal to Ø or small numbers. Our listing program can limit the printout, if necessary, to finds with a particular range of coordinates or lengths.

Table I: Intermediate file of search finds for Figure 3.

X	Y	Length	Matches	Diagonal #	Rel. Phase
79	1	9	7	78	-
68	23	9	7	45	33
114	75	10	7	39	6
68	31	9	7	37	2
60	26	9	7	34	3
116	82	9	7	34	0
80	62	9	7	18	16
85	67	12	9	18	0
90	72	13	11	18	0
96	78	12	9	18	0
39	29	9	7	11	7
42	31	9	7	11	0
52	41	9	7	11	0
54	43	18	14	11	0
74	74	9	7	0	11
40	67	10	7	-27	27

Plotting Techniques.

Rotating the Plot Display. A key feature of the program which performs the plotting function is that the picture is rotated forty-five degrees counter-clockwise from the traditional perspective. While this may look a little strange as it is being generated, there are several justifications for doing it this way. Most basically, this is the order in which the homologous segment finds are reported out of the search program. We have been using as a plotting device a printer which can only reliably roll the page forward. This being the case, generating the plot without the rotation would require an intermediate step to reorder the data.

The rotation greatly improves the ability to minimize the motion (and hence time) required to produce the plot on many devices (including ours.) This will be the case for all plotting devices where line segments can not be drawn down a diagonal on the page as easily as those drawn horizontally. This is significant to the extent that the plot contains sets of homologous segments nearly in global alignment with one another. Such segments are, after rotation, displayed one after the other across the page or screen.

Another advantage of rotating the plot is seen on all discrete plotting devices, i.e., those that generate the image

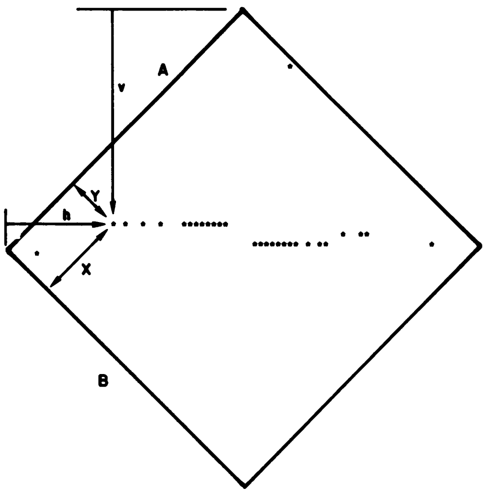

Figure 5. A dotplot produced by a non-graphics printer. Sequence A is polyoma virus; Sequence B is SV40. The stringency is 40/70. The plot is oriented as it was during printing. The position and length of lines are calculated as follows. S is a scaling factor such that S(length(A)+length(B)) = the available width of the plotting area. Horizontal position (h) = S(X+Y). Vertical position (v) = S(length(A)-X+Y). Plotted line length = 2S times the reported line length.

from a set of dots. For this class of plotting device, line segments represented diagonally on the plot often appear as a jagged line. This is undesirable since because gaps in the sequence alignment produce the same effect. Our routine eliminates this problem by lining up the diagonals with the horizontal axis of the printer.

Displaying Entire Segments. Line segments are drawn on our plots to represent the entire length of the homologous segments, not just their central element. We feel this representation is more accurate and yields more easily recognized homology features. These segments are produced as a continuous line (to the maximum horizontal resolution) independently of the number of positions within the corresponding sequence segments. Our ability to generate these segments is made much easier (again, on most devices) due to the rotational transformation.

Scaling. The size of the plot may be varied independently of the length of the sequences represented or the size of the physical plotting surface. Very large plots may be generated in strips. For those plots intended for publication we have found that by scaling the dimensions down to the approximate final size, we avoid the problem of reducing line widths below that easily seen within figures. Tic marks are generated around the boundary to give a frame of reference sufficient to locate the coordinates of the interior segments.

Plotting on Non-graphics Equipment. Most of the dotplots in this paper were generated on a printer with 1/120th inch horizontal and 1/60th inch vertical resolution. Figure 5 shows the same polyoma/SV40 comparison as in Figure 1 printed at 1/10 inch horizontal and 1/6 inch vertical resolution, totally without the use of graphics capabilities. The plot is shown oriented as it was during printing. Even though the resolution is considerably less, the same major features are readily apparent.

Implementation.

It has been our intention to define the algorithms so as to make it easy for programmers to rewrite them for use in systems much different than ours. Our own programs will be made available upon request, probably through the Unix network. (Unix is a trademark of Bell Laboratories.) They are applicable to a Z-80 based CP/M system. (CP/M is a trademark of Digital Research.) The sequence files are generated with a sequence editor written in Pascal/Z, named SED. The search program, named DIAGSRCH, was written in Microsoft Fortran and has the search routine itself written in assembly language. The listing program, DIAGLIST, is written in Fortran. The plotter program, DIAGPLOT, is written in the language C to drive a DIABLO 1620 printer in graphics mode. The plot in Figure 5 was made with a small demonstration program written in BASIC, called CRUDEPLOT.

ACKNOWLEDGEMENTS

This research was supported by Public Health Service Grants AI08998 and GM21313 from the National Institutes of Health. S.C.H was supported by an NIH fellowship.

REFERENCES

1. Gibbs, A.J. and McIntyre, G.A. (1970) Eur. J. Biochem. 16, 1-11.
2. McLachlan, A.D. (1971) J. Mol. Biol. 61, 409-424.

3. Maizel, J.V. Jr. and Lenk, R.P. (1981) Proc. Nat. Acad. Sci. USA 78, 7665-7669.
4. Steinmetz, M., Frelinger, J.G., Fisher, D., Hunkapiller, T., Pereira, D., Weissman, S.M., Uehara, H., Nathenson, S. and Hood, L. (1981) Cell 24, 125-134.
5. Fristensky, B., Lis, J. and Wu, R. (1982) Nucl. Acids Res. 10, 6451-6463.
6. Harr, R., Hagblom, P., and Gustafsson (1982) Nucl. Acids Res. 10, 365-374.
7. Jagadeeswaran, P. and McGuire, P.M. Jr. (1982) Nucl. Acids Res. 10, 433-447.
8. Novotny, J. (1982) Nucl. Acids Res. 10, 127-131.
9. Pustell, J. and Kafatos, F.C. (1982). Nucl. Acids Res. 10, 4765-4782.
10. Staden, R. (1982) Nucl. Acids Res. 10, 2951-2961.
11. Konkel, D.A., Maizel, J.V. Jr. and Leder, P. (1979) Cell 18, 865-873.
12. Edgell, M.H., Weaver, S., Jahn, C.L., Padgett, R.W., Phillips, S.J., Voliva, C.F., Comer, M.B., Hardies, S.C., Haigwood, N.L., Langley, C.H., Racine, R.R. and Hutchison, C.A. III (1981) in Organization and Expression of Globin Genes, Stamatoyannopoulos, G. and Nienhius, A.W. Eds., pp.69-88, Alan R. Liss, Inc., New York.
13. Max, E.E., Maizel, J.V. Jr. and Leder, P. (1981). J. Biol. Chem. 256, 5116-5120.
14. Jeffreys, A.J., Barrie, P.A., Harris, S., Fawcett, D.H., Nugent, Z.J. and Boyd, A.C. (1982) J. Mol. Biol. 156, 487-503.
15. Moschonas, N., de Boer, E. and Flavell, R.A. (1982) Nucl. Acids Res. 10, 2109-2120.
16. Hardies, S.C., Edgell, M.H., and Hutchison, C.A. III (1983) submitted for publication.
17. Fitch, W.M. (1969) Biochemical Genetics 3, 99-108.
18. Needlemann, S.B. and Wunsch, C.D. (1970) J. Mol. Biol. 48, 443-453.
19. Fitch, W.M. and Smith, T.F. (1983) Proc. Nat. Acad. Sci. USA 80, 1382-1386.
20. Reddy, V.B., Thimmappaya, B., Dhar, R., Subramanian, K.N., Zain, B.S., Pan , J., Ghosh, P.K., Celma, M.L. and Weissman, S.M. (1978) Science 200, 494-502.
21. Soeda, E., Arrand, J.R., Smolar, N., Walsh, J.E. and Griffin, B.E. (1980) Nature 283, 445-453.
22. Spritz, R.A., DeRiel, J.K., Forget, B.G. and Weissman, S.M. (1980) Cell 21, 639-646
23. Phillips, S.J., Hardies, S.C., Jahn, C.L., Edgell, M.H. and Hutchison, C.A. III (1983) submitted for publication.
24. Lawn, R.M., Efstratiadis, A., O'Connell, C. and Maniatis, T. (1980) Cell 21, 647-651.
25. Hutchison, C.A. III, manuscript in preparation.