Volume 14 Number 1 1986 Nucleic Acids Research

Improving the efficiency of dot-matrix similarity searches through use of an oligomer table

Brian Fristensky

Department of Plant Pathology and Program in Genetics and Cell Biology Washington State University, Pullman, WA 99164-6430, USA

Received 17 July 1984 Revised 22 April 1985 Accepted 8 May 1985

ABSTRACT

Dot-matrix sequence similarity searches can be greatly speeded up through use of a table listing all locations of short oligomers in one of the sequences to find potential similarities with a second sequence. The algorithm described finds similarities between two sequences of lengths M and N, comparing L residues at a time, with an efficiency of

$$L \times M \times N/(S^k)$$

where S is the alphabet size, and k is the length of the oligomer. For nucleic acids, in which S=4, use of a tetranucleotide table results in an efficiency of L x M x N/256. The simplicity of the approach allows for a straightforward calculation of the level of similarities expected to be found for given search parameters. Furthermore, the storage required is minimal, allowing for even large sequences to be compared on small microcomputers. Theoretical considerations regarding the use of this search are discussed.

INTRODUCTION

The dot-matrix similarity search algorithm [1] is probably the most widely-used method for comparing two nucleic acid or protein sequences. It can easily be conceptualized by imagining two sequences, X and Y, being placed on the X and Y axes of a matrix. A search program would then move down the Y axis, comparing each substring of L contiguous characters in Y with all possible substrings

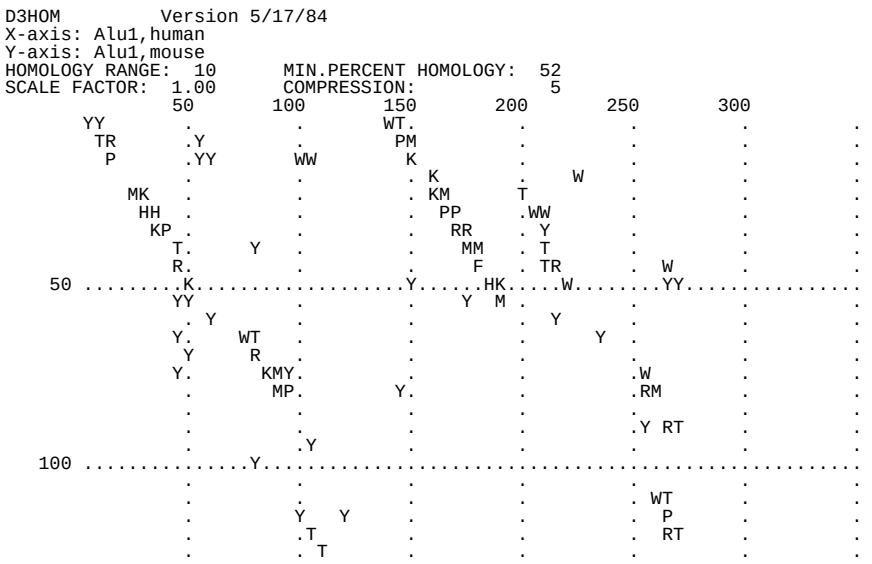

Fig.1. Similarity comparison between two Alu1 Family sequences. The sequence on the X-axis is the human Alu1 Family consensus sequence [2] and the Y-axis sequence is the mouse consensus [3]. The output is from D3HOM, described below. Characters in the matrix indicate similarities found between the two sequences which share greater than 52% match. The similarity range (w in the text) is 10, which means that 10 bases on either side of the central match were compared for each local similarity. Using the conventions of Pustell & Kafatos [4], each character indicates the level of the similarity, in descending increments of 2% going from A to Z. Thus, an A in the matrix would represent 100% match (21/21), B= 98-99% match, ... Y=52-53% match (11/21) etc. The presence of two diagonals indicates that the human Alu sequence is an imperfect tandem repeat of the original ancestral sequence which is only a monomer in the mouse.

of L characters in X. If a match is found which is good enough, based on some pre-defined criterion, then a symbol (eg. a dot) is printed at the corresponding X,Y position in the matrix. Where long stretches of similarity occur between the two sequences, a diagonal line of symbols will be seen in the matrix (Fig.1).

A quick inspection of most dot-matrix similarity outputs shows that the vast majority of the area of the matrix contains either blank space, which indicates that no local similarities were found, or very small similarities which have probably occurred at random. Thus, the majority of the search time is spent investigating non-similar regions. Inspection of similar sequences leads to the observation that sequences which are recognizable as being similar share many

contiguous matches. Martinez, for example, has exploited this property of similarities for the purpose of finding perfect repeats within sequences [5]. Although his algorithm is very fast, it is of little practical value, since it will not find imperfect similarities. Wilbur and Lipman [6] have recognized the fact that even imperfect similarities are likely to share small regions of perfect similarity (eg. 4 bases). They have taken the approach of pre-screening both sequences for perfect matches, which can be used as starting points in building a best-fit similarity of the type first described by Needleman and Wunsch [7]. (Algorithms of this type will be referred to as NWS algorithms, following the convention of Goad and Kanehisa [8].)

The dot-matrix similarity algorithm, while not producing an "optimal" alignment, instead presents the user with an over-all view of all similarities between two sequences. It is usually an exhaustive search, in the sense that every possible comparison between the two sequences is made. This paper presents a non-exhaustive dot-matrix search algorithm which uses a table of the locations of k-mers in one sequence to find potential regions of similarity with a second sequence. For example, a table listing the locations of each of the 64 possible trinucleotides found in sequence X can serve as a guide for finding regions in sequence X which could share significant similarity with the subsequence from Y which is being sought.

THE ALGORITHM

I shall describe the algorithm in simple terms, using the case of a nucleic acid sequence comparison in which k=3. Obviously, a similar search could compare protein sequences, or use different k values. A more rigorously defined statement of the algorithm is given in Appendix I.

The first step involves construction of a table listing each occurrence of the 64 trinucleotides in sequence X (Table 1). The table will contain N-2 numbers representing the locations of the trinucleotides, divided among 64 classes. The search begins at the first trinucleotide in sequence Y. Using the table as a guide, each occurrence of that trinucleotide in sequence X is located, and the region centered on that posi tion, w nucleotides to the left and the right, is compared with the corresponding region in sequence Y. If the match is good enough, a symbol is printed at the point in

Table 1.

Trinucleotide	Location(s) in seq.X
AAA	13, 71, 179, 204, ...
AAC	35, 72, 123, 199, ...
AAG	7, 50, 87, 104, 249, ...
...	.....
...	.....
TTG	2, 40, 95, 172, ...
TTT	77, 94, 169, 195, ...

Table 1. Locations of the 64 possible trinucleotides in sequence X. The numbers shown indicate the position of the central nucleotide in a triplet, as they might occur in some hypothetical DNA sequence.

the matrix which corresponds to the centers of the two regions. The process is repeated for each trinucleotide in sequence Y. Since each trinucleotide occurs on the average only once every 64 bases, the algorithm only makes N/64 searches for each triplet in Y, rather than N.

Implementation

The similarity search algorithm has been implemented in four programs: P1HOM,P2HOM,D3HOM, and D4HOM. D3HOM compares two DNA sequences as described above, using a trinucleotide table. A typical plot is shown in Fig1. D4HOM is essentially the same as D3HOM, except that it uses a tetranucleotide table, which makes it four times faster. P1HOM and P2HOM perform protein sequence comparisons using mono- and dipeptide tables, respectively.

All programs were written in Standard Pascal [9], which is a subset of all implementations of Pascal. Non- standard or machine-dependent constructs were avoided. Consequently, they should run with only trivial changes on any computer for which Pascal is available. The programs were initially implemented on an AppleII+ computer under the UCSD Pascal system, and have also been run using the IBM (Microsoft) 1.0 compiler on an IBM PC and using IBM Pascal/VS in the VM 370/CMS environment. These programs are intended to augment the Cornell Sequence Analysis Package [10].

Choice of working parameters

The most important factor affecting the efficiency of the algorithm is the parameter k, the size of the oligomers used in the search table. Since the speed of the search increases with S^k, one might be tempted to use large k. However, since k represents the number of contiguous matches that must exist in order for a local similarity to be found, large k values will cause the algorithm to miss imperfect similarities.

Ideally, one would like to choose the maximum k such that significant local similarities will still be found. Although the definition will vary with the problem, one good way of defining "significant" is as a deviation from randomness. In other words, we are not interested in similarities so poor that they could have occurred by chance alone.

The probability p that a base or amino acid chosen independently from sequence X will match a base or amino acid chosen independently from sequence Y is given by

Eq. 1 $$p = \sum_{i=j} f_i f_j$$

where i and j are indicies representing either the four nucleotides or the twenty amino acids, and f_i and f_j are the frequencies with which the ith or jth bases or amino acids occur in sequence X and sequence Y, respectively. Thus, when comparing two DNA sequences with uniform base compositions ($f_A=f_C=f_G=f_T=0.25$), the probability of any two bases matching is 0.25. Similarly, for proteins of even amino acid composition, the probability of a given pair of amino acids matching is 0.05. The actual base compositions of two sequences being compared will affect the probability p that any two bases will match.

The probability that two k-mers chosen at random will match is simply p^k. The expected distance between two occurrences of k matches is therefore $1/p^k$. To insure a thorough search, we must choose a combination of k value and window size such that the region L bases wide which is searched at one k-mer match will overlap the adjacent window. The average distances between k-matches for different values of p and k are given in Table 2. For example, if the probability of a match between two DNA sequences is 0.25, we expect to see a dinucleotide match once every 16 bases in a comparison. Trinucleotides will match on the average of once every 64 bases, and so on. These matches are due to background similarity. Since regions which share significant similarity must by definition have a frequency of matches which is higher than background, similar regions will have more frequent k-mer matches, and consequently are more likely to be found.

Table 2 illustrates how the overall level of similarity between two sequences affects the expected distance between k-mer matches. The knowledge of the expected frequency of k-mer matches allows us to predict the level of similarity likely to be missed. If we wish to find similarities with 30% match or better, a triplet search (k=3) will

Table 2.

Prob. of a match P	Avg. dist. between k-matches $\frac{1}{P^k}$			
k=	2	3	4	5
0.050	400	8000		
0.075	178	2370		
0.100	100	1000		
0.150	44	296		
0.200	25	125		
0.250	16	64	256	1024
0.300	11	37	123	412
0.350	8	23	67	190
0.450	5	11	24	54
0.600	3	5	8	13
0.700	2	3	4	6
0.900	1	1	1	2

Table 2. Average distance between occurrence of k contiguous matches as a function of the probabilty of a single match. Calculations were done for a range of probabilities and reasonable k values. Distance values are rounded to the nearest integer.

necessitate the use of a window size w >= 19, since the average distance between triplet matches is 37 (see Table 2). The actual choice of k and w values will depend on the purpose of the search. For example, if one wished only to identify overlapping regions of DNA subclones sequenced by the shotgun strategy, then it would be perfectly appropriate to use a 5-mer search, since p should be very high (presumably, we're comparing identical stretches of DNA) and the expected distance between pentanucleotide matches is 2, even if only 90% of the sequence was correct. If comparing two sequences which hybridize only under low stringency conditions, we still expect no worse than about 60% similarity. For this purpose, a k value of 4 and a window size of 5 would be sufficient to insure that hybridizing sequences were found. For poorer similarities which approach random background, a 3-mer search would be necessary.

The expected distances between k-mer matches given in Table 2 were derived under the assumption that the probability of a k-mer match at a given point on the diagonal is independant of the occurrence of other k-mer matches. However, overlapping k-mer matches will occur whenever a run of matches greater than k is found on a given diagonal. For example,

if a trinucleotide match is found at position (X,Y) in the matrix, the probability that a trinucleotide match will occur at (X+1,Y+1) is p, the probability of a single match, rather than p^3. This is because the trinucleotide match at (X+1,Y+1) overlaps two of the single matches in the previous trinucleotide.

If we assume that k-mer matches occur according to a binomial probability distribution, then the probability P_d that the next k-mer match will occur within d bases of a given k-mer match is

Eq. 2
$$P_d = \sum_{i=1}^{k} p^i + \sum_{i=k+1}^{d} p^k (1-p^k)^{i-k}$$

where the left-hand summation represents the cumulative probability that the next k-mer match overlaps the previous one, and the right-hand summation represents the cumulative probability of k-mer matches beyond the overlap. Eq.2 can be transformed into an easier to evaluate form,

Eq. 3
$$P_d = (1-p^k)\left[\frac{p}{1-p} + 1 - (1-p^k)^{d-k}\right]$$

Figure 2 illustrates that P_d increases most rapidly at low d values, ie. within the first few bases of a given k-mer match. This is due to overlap, and leads to the prediction that k-mer matches will tend to cluster. To find the distance by which a specified fraction (P_d) of k-mer matches are expected to be found, Eq.3 can be solved for d, to give

Eq. 4
$$d = \frac{\ln\left[1 - \frac{P_d}{(1-p^k)} + \frac{p}{1-p}\right]}{\ln(1-p^k)} + k$$

In other words, d is the distance from a given match at which the probability of finding at least one k-match is P_d. Thus, if we wish to approach certainty of finding at least one tetranucleotide match (P_d = 1.0) and the probability of a single base match p is 0.40, then we must set the size of the search window to d > 21. This is much lower than the value which would be obtained using $d = 1/p^k = 39$. Significantly, if

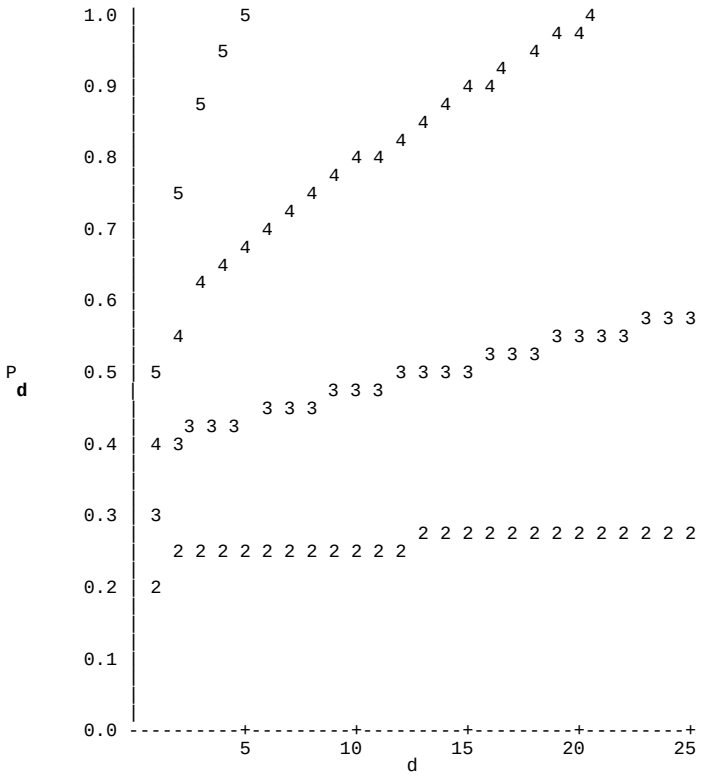

Fig.2. P_d for tetranucleotide matches (k=4) as a function of d, for different p values. Equation 2 was evaluated using p=0.2 (represented by 2 in the graph), p=0.3 (3), p=0.4 (4), and p=0.5 (5).

we only demand that 75% of the matches be found, d need only be greater than 8. It appears, then, that use of the simpler calculation shown in Table 2 is actually an overly conservative method for calculating search window sizes, due to clustering of matches.

DISCUSSION

Efficiency

I have described a modification of the dot- matrix similarity search algorithm which improves the efficiency of the standard algorithm by decreasing the amount of time spent comparing regions which are not likely to share significant similarity. Since the time required to create the k-mer table is quite short (on the order of the length of

the X-axis sequence), only the search loop itself needs to be considered. At each of M-(L-1) positions in sequence Y, a region of L characters is compared with each of, on the average, N-(L- 1)/(S^k) regions of L characters in sequence X. Since L is usually quite small relative to N and M, the efficiency of the algorithm can be expressed as

$$L \times M \times N/(S^k).$$

When the probability of finding random local similarities is high, or the size of L is comparable to S^k, this algorithm will approximate to M x N. As in most similarity searches, the slowest execution will be in areas in which strong similarities exist.

Storage

Aside from the sequences themselves, which use M + N units of memory (i.e. bytes), the only major storage requirement of the algorithm described here is for the table of linked lists used in the search. There are N-(k-1) k-mers in sequence X, divided among S^k elements in a k-dimensional table, which is defined over an alphabet of size S. In order to allow for the presence of unknown nucleotides (N's) or amino acids (X's) in the sequences, the table must actually contain $(S+1)^k$ elements, so that elements with one or more subscripts being N (or X) will be set to nil. Table elements which have no unknowns will point to a linked list of all occurrences of the corresponding k-mer, or nil if that k-mer did not occur in sequence X. Finally, each node of the linked list requires two fields, a position field and a pointer to the next node, each of which will probably require two bytes. Thus, the storage required for the table is $(S+1)^k$ + 4(N-(k-1)). A DNA sequence of 1000 bases will require about 4117 bytes to be encoded in a triplet table.

For larger sequences, the upper limit to the size of the table is not memory (except on some microcomputers), but the width of the output page. For example, if the graph is compressed twentyfold, then each dot or character in the matrix represents 400 nucleotide comparisons. Even if the program is set up to print only the best similarity which occurred in a given place, as in my programs and the Pustell programs, the background will begin to overwhelm the signal when very high compressions are used. Consequently, only part of the X-axis sequence (eg. <=3000) will participate in the search at any given time, and it is only for that part that the table needs to be constructed. Finally, it should be noted that the efficiency of the algorithm is the same regardless of which sequence is indexed in the table. Thus the memory

used can be minimized by placing the shortest sequence on the X-axis.

Sensitivity

Because the algorithm described in this paper does not make every possible comparison between two sequences, it may miss some local similarities. As with all other similarity search algorithms, long regions of strong similarity are easy to find. Conversely, the most difficult similarities to find are the poorest ones, and there is no single approach best suited to finding them.

Even exhaustive searches can miss significant similarities. For example, the heuristic search algorithm of Korn & Queen [11] builds similarities until they can no longer be extended and then decides whether they are significant enough to be printed, based on user-defined criteria. The main drawback of this approach is that long stretches which contain discontinuous regions of similarity appear as fragmented, short similarities in the output. These often go unrecognized by the user due to the fact that the "significant" similarities are interspersed among a long listing of presumably random similarities.

NWS-type algorithms are usually exhaustive in the sense that all possible combinations of matches and gaps are considered in generating locally optimal alignments. (One exception is the Wilbur-Lipman algorithm, which is non-exhaustive, and may miss similarities in which large gaps occur.) In the case of poor similarities, there may be several equally good alignments, and the best alignment may not always be the most "significant" one. Programs which present alternative alignments in a non-graphic form [6,12] again leave the user with the problem of identifying significant similarities and of connecting local similarities separated by gaps. Finally, all NWS-type algorithms require the user to set an arbitrarily chosen penalty value for gaps. Since the generation of gaps during evolution is due to many different biological mechanisms (insertion, deletion, transposition, unequal crossing over, geneconversion, DNA-repair errors, chromosomal aberrations etc.) the results may be misleading.

One great advantage of the non-exhaustive search algorithm described in this paper is that it lends itself to the easy calculation of the levels of similarities likely to be missed using given search parameters, as described above. For example, in a 4-mer search with a window size of 12, the regions sampled will not overlap if the distances between their centers are greater than 24. Checking Table 2, we find that the average distance between tetranucleotide matches is 24 when the

overall similarity is 45%. Therefore, under these search conditions, we expect to miss many similarities of 45% or less. As illustrated by equation 4 and Fig.2, however, the search windows calculated in this manner are actually a conservative approximation, since the overlap of adjacent k-mer matches will result in a lower average distance between k-mer matches. Thus, use of a large enough search window and a low enough minimum match insures that most sequences which share any significant similarity will be compared. Although worst case similarities can be hypothesized in which high levels of similarity occur with no k-mer matches, they do not represent a significant proportion of the theoretically possible cases.

The background of random similarities found is higher than one might expect. Lipman et al. [13] have used Monte Carlo simulations to demonstrate that similarities in overall base composition, nearest neighbor frequencies, or local base composition can result in statistically significant similarities even among randomly generated sequences. Others have shown that sequences which are related in function (eg. protein coding regions [14,15] and intervening sequences [15]) share certain statistical properties with respect to base composition, nucleotide preference within codons, nearest neighbor frequencies, and strand asymmetry. Hence, while there are many ways of quantifying the statistical significance of a match, it is much harder to evaluate its biological significance. It is in this borderline area, in which statistical significance and biological significance do not always agree, that non-exhaustive algorithms will miss local similarities.

Applications

The k-mer search described in this paper seems to be applicable for essentially any nucleic acid similarity comparison, provided that the right search parameters are used. Unfortunately, however, it will probably not be very useful for database searches, since the amount of printed output would be enormous. For example, using a compression of 20 characters per nucleotide, a dot-matrix search of a 3 million nucleotide database would produce 150,000 lines of output. For smaller subsets of databases, use of a dot-matrix search may be feasable, and perhaps preferable to non-graphic searches since they can be scanned rapidly by eye.

Protein similarity searches present another set of problems. Although for a given k size the protein search will be much faster

(since S=20), poor similarities are more likely to be missed because the statistical likelihood of a match is lower. As is shown in Table 2, the average distance between dipeptide matches is 400 (assuming uniform amino acid composition). This is longer than the length of many proteins. Thus, we can not depend on random chance to cause the algorithm to sample the two sequences frequently in regions of low similarity. While good similarities are likely to be found by the dipeptide search, some very poor but statistically significant ones will not. For cases in which such poor similarities must be found, a monopeptide search (eg. P1HOM) can still speed up the search, since it will have an efficiency 20 times faster than the L x M x N algorithm. The monopeptide search can be considered exhaustive because it only requires single matches. (Obviously, all similarities must have at least single matches.)

Conclusion

No single search algorithm is sufficient or appropriate for all applications. As discussed above, even exhaustive searches can miss significant similarities, either by nature of the search itself, the parameters used, or by the way in which the data are presented to the user. Paradoxically, the strong points of the different algorithms are usually also their weak points. While NWS-type algorithms are needed to produce the best-fit similarities necessary for quantifying evolutionary distance, the answers they give are not always the "correct" ones, and they may cause important features of the overall similarity picture to be overlooked. Finally, the optimal alignment will often include internal regions which share little similarity but have been forced into a fit. The advantage of dot-matrix searches is that much more data is presented to the user, since all local similarities above a certain level are printed. The program itself makes no decisions with regards to gaps; these are manifested to the eye as displacments of the diagonal. The pattern recognition abilities of the human brain allow for almost instant identification of the important features in a similarity plot. At the same time, human pattern recognition abilities must be considered as rather subjective and only semiquantitative. Consequently, the best strategy for similarity searches is probably to do the initial searches using a dot-matrix program, and then optimize the similarity using an NWS search.

ACKNOWLEDGEMENTS

I would like to thank Michael Moody, Thomas S. Russell, and Minoru Kanehisa for helpful discussions and Catherine Daniels for comments and criticism of the manuscript. This work was supported in

part by NSF Grant PCM-8203176 and University of Washington Sea Grant No. RX-10, both to Lee A. Hadwiger. Scientific Paper No.SP6889, Project 1844, College of Agriculture and Home Economics, Washington State University.

NOTE During the preparation of this manuscript, matrix similarity programs using an algorithm similar to that described in this paper were written by James Pustell.

REFERENCES

1. Maizel, J. and Lenk, R.P. (1981) Proc. Natl. Acad. Sci. USA **78**, 7665-7669.
2. Deininger, P.L., Jolly, D.J., Rubin, C.M., Friedmann, T. and Schmid, C. (1981) J.Mol.Biol. **151**, 17-33.
3. Krayev, A.S., Kramerov, D.A., Skryabin, K.G., Ryskov, A.P., Bayev, A.A. and Georgiev, G.P. (1980) Nucl. Acids Res. **8**, 1201-1215.
4. Pustell, J., and Kafatos, F. (1982) Nucl. Acids Res. **10**, 4765-4782.
5. Martinez, H.M. (1983) Nucl. Acids. Res. **11**, 4629-4634.
6. Wilbur, W.J., and Lipman, D.J. (1983) Proc. Natl. Acad. Sci. **80**, 726-730.
7. Needleman, S.B., and Wunsch, C.D. (1970) J. Mol. Biol. **48**, 443-453.
8. Goad, W. and Kanehisa, M. (1982) Nucl. Acids Res. **10**, 247-263.
9. Jensen, K., and Wirth, N. (1974) Pascal User Manual and Report, 2nd Ed., Springer Verlag, New York.
10. Fristensky, B., Lis, J., and Wu, R., (1982) Nucl.Acids Res. **10**, 6451-6463.
11. Korn, L.J., Queen, C.L., and Wegman, M.N. (1977) Proc. Natl. Acad. Sci. USA **74**, 4401-4405.
12. Taylor,P. (1984) Nucl. Acids Res. **12**, 447-455.
13. Lipman,D.J., Wilbur, W.J., Smith, T.F., and Waterman, M.S, (1984) Nucl.Acids Res. **12**, 215-226.
14. Fickett, J. (1982) Nucl. Acids Res. **10**, 5303-5318.
15. Smith, T., Waterman, M.S. and Sadler, J.R. (1983) Nucl. Acids Res. **11**, 2205-2220.

APPENDIX I - FORMAL DEFINITION OF THE ALGORITHM

The sequences are represented by two arrays, SEQX and SEQY, where SEQX[1],...,SEQX[N] represents sequence X, while SEQY[1],...,SEQY[M] represents sequence Y. The parameter w defines the number of nucleotides compared on either side of the central base. Thus, if w = 10 then a 21 base region will be compared at each position. If the sequences are linear, SEQX[-w],...,SEQX[0] and SEQX[N+1],...,SEQX[N+w] are set to Z, and SEQY[-w],...,SEQY[0] and SEQY[M+1],...,SEQY[M+w] are set to N. The unknown nucleotides N and Z do not match, so other parts of the sequence containing unknowns will not match anything. This is necessary to prevent the discovery of false matches. For circular sequences, SEQX[-w],...,SEQX[0] is initialized to SEQX[N-w],...,SEQX[N], and SEQX[N+1],...,SEQX[N+w] is initialized to SEQX[1],...,SEQX[w]. Similarly, SEQY[-w],...,SEQY[0] is initialized to SEQY[M- w],...,SEQY[M], and SEQY[M+1],...,SEQY[M+w] is initialized to SEQY[1],...,SEQY[w].

The trinucleotide table is stored as a 3-dimensional array TRIPLET[n1,n2,n3] whose subscripts may be A,C,G,T, or N, but not Z. Each element of TRIPLET is a pointer to the first element in a linked-

list of nodes, each of which holds an integer POS, which gives the position of one occurrence of the trinucleotide, and NEXT, which points to the next node in the list, or nil. Thus, using Table 1 as an example, TRIPLET[A,A,C] points to a list of all occurrences of AAC in SEQX, the first of which is at position 35. For each single base match found in a local similarity, the variable SCORE is incremented by a value from the array SUBSCORE, such that SUBSCORE[i] is the value to be added to SCORE for a match i bases from the center of the triplet. As in the programs of Pustell [4],

$$SUBSCORE[i] = vf^i$$

where v is the value for a match at the center of the triplet (at which i=0) and $0 < f < 1$. MINSCORE is the minimum score for which a similarity will be recorded in the matrix. If f=1, then the score is proportional to the percentage of the total comparisons made. If $f < 1$, then the contribution of a match decreases exponentially as a function of its distance from the center of the triplet. As Pustell and others have shown, weighting of matches helps decrease background similarities.

The algorithm to compare the portion of SEQY from STARTY to FINISHY with the portion of SEQX from STARTX to FINISHX is programmed as follows:

```
MAKETABLE(STARTX,FINISHX); {make trinucleotide table of SEQX}
for Y:= STARTY to FINISHY do
  SEARCHFOR(TRIPLET[SEQY[Y-1],SEQY[Y],SEQY[Y+1]])
```

where the procedure SEARCHFOR is defined as follows:

```
  procedure SEARCHFOR(TRI); {TRI points to 1st occurrence of triplet in SEQX}
    begin
      while TRI <> nil do begin
        X:= TRI^.POS; {Location of a trinucleotide in seq.X}
        SCORE:= THREEMATCH; {Score for central triplet match}
        LX:= X-2; RX:= X+2;{Begin comparison at 2nd posn.from center}
        LY:= Y-2; RY:= Y+2;
        DISTANCE:=2; {Dist. from center of triplet}
        while DISTANCE <= w do begin
          if SEQX[LX] = SEQY[LY] then SCORE:= SCORE+SUBSCORE[DISTANCE];
          if SEQX[RX] = SEQY[RY] then SCORE:= SCORE+SUBSCORE[DISTANCE];
          LX:= LX-1; RX:= RX+1; LY:= LY-1; RY:= RY+1;
          DISTANCE:= DISTANCE+1
          end;
        if SCORE >= MINSCORE then print a symbol in the matrix;
        TRI:= TRI^.NEXT {move to next posn. in list, if there is one}
        end
    end { SEARCHFOR }
```