Algorithms

The following is only a brief description of the algorithms used in MacPattern. For detailed information please refer to the cited literature.

Pattern Matching Algorithm

The pattern matching algorithm employed by MacPattern v3.0 and higher is more than twice as fast than the one used in earlier versions. Most of this speed improvement results from binary encoding the query sequence and the PROSITE patterns plus adopting a sequence hashing approach.

The encoding is performed by converting an amino acid ASCII code into a unique binary value; for example, Ala becomes 00000001, Gly 00000010, Cys 00000100, and so on. PROSITE patterns are converted to a table with one entry for each pattern position. Each table entry includes a “mask” value which, in binary format, reflects the list of allowed and excluded residues at this position, created by transforming the ASCII pattern description into a binary representation as described above and applying Boolean algebra.

Sliding this table along the sequence allows the identifiation of matching regions in a highly efficient manner, because only simple Boolean comparisons are required. To accommodate well-defined repetitions of residues at particular positions - for example, A(3,5) in PROSITE notation means 3, 4 or 5 alanines are allowed - the pattern table includes the minimal and maximal allowed number of occurrences of residues at each position. These values are considered during the matching step which is modified by introducing a depth-first search procedure at variable-length pattern positions.

This basic algorithm is speeded up furthermore by hashing the input sequence after encoding it into binary format. If there is some invariant residue position in the pattern, i. e. only one particular residue is allowed at that position, then this hash table approach allows the quick identification of potential match regions in the input sequence by rapidly locating all occurrences of the invariant residue. Matching is then only attempted around these anchor points. Thus, instead of sliding along the sequence one effectively “jumps” from anchor point to anchor point, resulting in a considerable reduction of the number of necessary comparisons.

Block Search Algorithm

The approximate block search algorithm used by MacPattern v3.0 and higher is described in detail elsewhere (Fuchs, 1993b). It is a modification of the basic method by Henikoff et al. (1990).

In brief, from a BLOCKS entry a site-specific scoring matrix is constructed, in which each

column is filled with values that reflect the frequency of occurrence of each amino acid at the coresponding position of the block. For the computation of a scoring matrix column, the arrangement of very closely related proteins into subgroups within a BLOCKS entry is taken into account and frequencies are averaged for each subgroup to reduce the contribution of multiply represented subfamilies (Henikoff et al., 1990). As from BLOCKS Release 7, the explicit weights supplied for each sequence are used instead. Furthermore, the resulting values are then adjusted to compensate for database bias by dividing them by the frequency of this amino acid in SWISS-PROT. The values in each column of the matrix are finally normalized to 100%.

The resulting matrix is used to calculate a score for each fragment of the query sequence of the width of the block by adding the matrix values for each residue at each position, and the highest scoring fragment is identified. To allow comparison of scores obtained with blocks of different length, a raw score is adjusted by dividing it by the "99.5%" value of this block, which is the 99.5th percentile of raw scores of true negative sequences with this block, and multiplying it by 1000 (Wallace and Henikoff, 1992).

This basic method has been modified by employing a hashing technique, similar to the one used for pattern searches. Invariant positions within a block are identified and serve as anchor points to jump along the query sequence (Fuchs, 1993b). This approach is 2 to 5 times faster than the standard method, however, it yields only approximate results and may fail to identify some biologically significant similarities. Therefore, by providing an adjustable "sensitivity" parameter, MacPattern gives complete control to the user who can decide to what extend she wants to make use of this option. Invariant positions may occur simply by chance, and they become more reliable the more sequences there are in a block. The sensitivity parameter is effectively the minimum number of sequences that must appear in a block before MacPattern applies the approximate algorithm. If the block features less sequences, the standard sliding-window approach is applied. The lower the threshold is the faster the search will be, but the chance of missing some hits increases as well.

The choice of the sensitivity parameter depends on the particular problem at hand: for a thorough analysis of individual sequences, one would choose "exact search", while for the rapid, preliminary analysis of a large number of sequences a low threshold may be perfectly appropriate. For routine application a value of 5 seems reasonable.

MMS Analysis Algorithm

Maximal segment score (MSS) analysis is based on a method introduced by Karlin and Altschul (1990). They showed that for sufficiently large sequences the probability of finding one or more distinct segments with scores greater than or equal to S is closely approximated by $1 - \exp(-KN \exp[-\lambda S])$, where N is the length of the sequence and the parameters K and λ are dependent on the scoring scheme and the letter frequencies. MacPattern uses the karlin() function from the BLAST program (Altschul et al., 1990) to compute K and λ for a given scoring scheme and input sequence. An algorithm similar to the one described by Karlin and Brendel (1992) is used to find variable-length fragments with maximum scores, and for each fragment found the probability of finding one with this score or higher is calculated.

Eguchi & Seto Algorithm

This method was proposed by Eguchi and Seto (1992). It is a variant of a family of k-tupel analysis algorithms (Claverie et al., 1990; Gabrielian et al., 1990). The basic assumption is that the most “dissimilar” oligomer in a sequence contains the highest information and is supposed to play a functionally or structurally important role. Eguchi and Seto could demonstrate that this method is well suited for the identification of active sites in peptide hormones.

For each amino acid in a sequence its average PAM score versus all other residues in the sequence is calculated. Then a window is slid along the sequence and the segment dissimilarity score is computed. The regions with lowest scores can then be identified.

The original method as proposed by Eguchi and Seto has been modified in MacPattern by using the BLOSUM62 similarity score matrix instead of PAM250, and by employing a triangular window smoothing technique (Claverie and Daulmerie, 1991) instead of a simple rectangular one, yielding much cleaner graphs.

While the usefulness of this method has been demonstrated for small proteins, its general applicability is unclear. Various tests with MacPattern have shown, however, that the hit rate (identification of a structurally or functionally important part of a sequence) is surprisingly high with general proteins.