BIOINFORMATICS

Vol. 17 no. 8 2001
Pages 676–685

Representation of amino acids as five-bit or three-bit patterns for filtering protein databases

Avril Coghlan[1], *Dónall A. Mac Dónaill*[2,*] *and Nigel H. Buttimore*[3]

[1]*Department of Genetics,* [2]*Department of Chemistry and* [3]*School of Mathematics, University of Dublin, Trinity College, Dublin 2, Ireland*

Received on February 12, 2001; revised on April 17, 2001; accepted on April 25, 2001

ABSTRACT

Motivation: We propose representing amino acids by bit-patterns so they may be used in a filter algorithm for similarity searches over protein databases, to rapidly eliminate non-homologous regions of database sequences. The filter algorithm would be based on dynamic programming optimization. It would have the advantage over previous filter algorithms that its substitution scoring function distinguishes between conservative and non-conservative amino acid substitutions.

Results: Simulated annealing was used to search for the best five-bit or three-bit patterns to represent amino acids, where similar amino acids were given similar bit-patterns. The similarity between amino acids was estimated from the BLOSUM45 matrix. Representing amino acids by these five-bit and three-bit patterns, the *Escherichia coli* PhoE precursor and the bacteriophage *PA2* LC precursor were aligned. The alignments were nearly the same as that obtained when BLOSUM45 was used to score substitutions.

Availability: The C code of the optimization algorithm for searching for the optimal bit-pattern representation of amino acids is available from the authors upon request.

Contact: dmcdonll@tcd.ie

1 INTRODUCTION

The problem tackled in this paper was clearly described by Myers (1995):

> Similarity searches over the ever-enlarging sequence databases are now routine, but the databases are growing so large that the time to conduct such searches is becoming problematic. A recently emerging theme to improve efficiency is to develop *filter* algorithms that eliminate a hopefully large percentage of the database that cannot contain a good match to a query sequence, and so leave one with a small subset upon which greater computational effort can be applied.

*To whom correspondence should be addressed.

A filter algorithm cuts down a database before running a second algorithm on the remainder of the database. The second algorithm, which Myers (1999) calls the *verification-capable* algorithm, is a more rigorous and time-consuming algorithm, such as the alignment algorithm of Smith and Waterman (1981). The filter algorithm must be faster than and as sensitive as the verification-capable algorithm, but not necessarily as selective. In bioinformatics two types of filtration are in use: *l-tuple filtration* and *filtration by composition*. The method of *l*-tuple filtration locates regions containing *l*-tuples (short exact matches) shared by a query sequence and a database sequence; the remainder of the database sequence is discarded (Dumas and Ninio, 1982; Pearson and Lipman, 1988; Pevzner and Waterman, 1995). Filtration by composition involves eliminating database sequence regions very different from the query in their one-tuple (Sibbald *et al.*, 1991), two-tuple (van Heel, 1991; Petrilli, 1993) or *l*-tuple composition (Blaisdell, 1989), or even in the mass spectrometry fingerprints of their enzymatic digestion products (James *et al.*, 1993).

We propose that a filtration method could be based on the dynamic programming optimization algorithm of Smith and Waterman (1981); what is new is that amino acids would be represented by five-bit or three-bit patterns, rather than by ASCII characters. We wish to emphasize that the aim of our work is not to speed up the dynamic programming algorithm, a problem that is not trivial (Green, 1996; Wozniak, 1997); but rather to represent amino acids in a new way, with possible implications for speeding up filter algorithms (or combinations of filter and verification-capable algorithms). At no point should this paper imply that we have the filtration algorithm. Instead it concentrates on the representation that would be needed for such an algorithm.

There are twenty amino acids, so five bits are needed for each to have a unique bit-pattern. However, three dimensions are enough to capture the basic mutational tendencies of residues (Taylor and Jones, 1993; Tomii and Kanehisa, 1996), so a three-bit representation of amino acids is also feasible.

Compared to previous filtration methods, ours would have four advantages. First, it would make full use of the wordsize of the computer, by packing five-bit or three-bit data into n-bit words, where n is commonly 32, 64 or 128. A three-bit representation of amino acids can be realised by assigning identical three-bit patterns $\boxed{abc}$ to the most closely related residues. Thus, on computers with a 32-bit wordsize, ten amino acids can fit into one word:

$$\boxed{b_{31}b_{30}|b_{29}b_{28}b_{27}|\cdots|b_5b_4b_3|b_2b_1b_0}$$

A *scoring function* that specifies the cost of substituting amino acid i for amino acid j is related to the XOR ($\oplus$) of the bit-patterns representing residues i and j (Buttimore and Mac Dónaill, 1996). The XOR results for bits a, b and c can be weighted by factors α, β and γ respectively, so that the cost of substituting amino acid i by amino acid j is:

$$\alpha(a_i \oplus a_j) + \beta(b_i \oplus b_j) + \gamma(c_i \oplus c_j).$$

Hence, by computing the XOR of two words, each of which represents ten amino acids, ten pairs of residues can be compared in one machine cycle. This *may* speed up calculation of the dynamic programming matrix central to the Smith–Waterman algorithm (1981), since previous Smith–Waterman versions compared just one pair of amino acids per machine cycle; this remains to be tested.

Second, the weighted XOR scoring function distinguishes between conservative and non-conservative amino acid substitutions. In contrast, l-tuple filtration and filtration by composition algorithms do not support conservative substitutions, simply scoring mismatches as 0 and matches as 1.

Third, an advantage of basing such a filter algorithm on the Smith–Waterman algorithm (1981) is that (like Smith–Waterman) it would allow for deletions and insertions, which would boost its sensitivity for detecting homologues. Both l-tuple filtration and filtration by composition algorithms search for short *ungapped* matches (l-tuples) so neither allow for insertions or deletions, except outside l-tuples (Pevzner and Waterman, 1995).

Fourth, our idea could be implemented as a cascade of progressively more stringent (and hence time-expensive) filters, as has been proposed for l-tuple filtration steps (Pevzner and Waterman, 1995). In the first step amino acids would be represented by three-bit patterns, and in the second step by five-bit patterns (Buttimore and Mac Dónaill, 1996). In other words, each amino acid sequence would be preprocessed to form a bit-vector of the three most highly weighted bits:

$$\boxed{b_{31}b_{30}|b_{29}b_{28}b_{27}|\cdots|b_5b_4b_3|b_2b_1b_0}$$

and a bit-vector of the other two bits:

$$\boxed{b_{31}b_{30}|b_{29}b_{28}|\cdots|b_3b_2|b_1b_0}$$

Representing each amino acid by three bits necessarily requires that some closely related amino acids be regarded as equivalent. A disadvantage of this approach could be a loss in sensitivity for detecting homologues. This tradeoff will allow faster initial comparisons. Moreover, we should note that since the XOR scoring function is restricted to taking a discrete set of values, the distances between 2^n n-bit patterns, it can only be an imperfect approximation of BLOSUM45.

Despite the speed advantage of comparing several pairs of residues in a single machine cycle, because the envisaged filter algorithm would be based on the time-expensive Smith–Waterman algorithm (1981), it would be slower than some other filters. That is not necessarily a disadvantage. For example, Pevzner and Waterman (1995) achieved an exponential reduction in the run-time of a verification-capable algorithm at the cost of linearly increased run-time of a filter algorithm. Hence, since verification-capable algorithms are more time-expensive than filter algorithms, it is worthwhile having a filter algorithm that is slower, but leaves a smaller subset of the database on which to run the verification-capable algorithm (that is, a filter that is more selective). Since the envisaged filter algorithm would allow for conservative substitutions, insertions, and deletions, it would be more selective than filters which do not, so the filter might compete well with others in terms of the combined run-time of the filter and verification-capable algorithms. That is, the extra cost in the filtration step may be offset by an overall saving in run-time.

The crux of the method would be that amino acids be assigned bit-patterns such that the similarity between bit-patterns is correlated with residues' mutational tendencies. There are approximately 5.5×10^{26} ways that five-bit patterns can be assigned to twenty amino acids, so this is a knotty problem. Indeed, if it took 0.001 s to assess each solution, checking all would take approximately 10^{16} years, or a million times the age of the Universe. Clearly we could not assess all binary representations of amino acids; rather we started with a reasonable mapping and began to optimize. This paper describes how we searched for the best bit-pattern to amino acid mapping, using an optimization algorithm.

The next section describes the optimization method employed. The third section explains how the optimization method was adapted to our problem. In the last section the best solution is tested and future steps to be taken are discussed.

2 METHODS

Simulated annealing was chosen as the optimization algorithm because it is easily implemented and can give good solutions in a short time (Tiourine *et al.*, 1995). This method is based on an analogy between

the physical process of annealing solids and the task of solving large combinatorial optimization problems. The developers of simulated annealing (Kirkpatrick *et al.*, 1983) were inspired by the Metropolis algorithm, which simulates the evolution of a solid in a heat bath to thermal equilibrium (Metropolis *et al.*, 1953). In the Metropolis algorithm, whenever a choice must be made between several configurations, the probability of changing from the present configuration having energy E_1 to a new one of higher energy E_2 is:

$$p = \exp[-(E_2 - E_1)/T]$$

where T, the *temperature*, is a constant (Press *et al.*, 1992). If $E_2 < E_1$ the system changes to the new configuration. In other words, increases in energy are occasionally accepted (with probability p), while decreases in energy are always accepted.

Thus, the Metropolis algorithm involves assessing N new configurations, using a constant value of the parameter T. *Simulated annealing* involves the Metropolis algorithm being run again and again, where each run involves assessing N new configurations. The value of T is constant during each such run, but at the start of each run T is decreased and so the probability (p) of accepting an increase in energy reduces.

To implement simulated annealing, one needs (Press *et al.*, 1992):

(1) A description of possible configurations, i.e. bit-pattern to amino acid mappings.

(2) A way of generating random changes in configuration; the new configurations generated are assessed and either accepted or rejected.

(3) An *objective function*, E, which is analogous to energy. The aim of the optimization algorithm is to to minimize E.

(4) A *control parameter*, T, which is analogous to temperature, and an *annealing schedule* that specifies how many configuration changes to assess before T is reduced (that is, it specifies N), and by how much to reduce T.

A configuration was defined by three sets of parameters (for example, see Table 1):

(1) Five floating-point weighting factors (α, β, γ, δ, ϵ), one for each of the five bits representing an amino acid (bits a, b, c, d, e).

(2) An arrangement of the thirty-two possible five-bit patterns in twenty sets.

(3) The assignment of a bit-pattern set to each amino acid.

If a five-bit pattern $\boxed{b_4b_3b_2b_1b_0}$ is interpreted as a *binary* number, then bits b_4, b_3, b_2, b_1 and b_0 must have weightings $\alpha = 2^4$, $\beta = 2^3$, $\gamma = 2^2$, $\delta = 2^1$, and $\epsilon = 2^0$, respectively. In contrast, we propose that the bit-patterns that represent amino acids are *not* restricted to being binary numbers. That is, α, β, γ, δ, and ϵ can take any values, not just powers of two.

As the initial configuration we used the bit-pattern to amino acid mapping previously proposed (Buttimore and Mac Dónaill, 1996; Table 1). This mapping was chosen assuming that the bits would be ordered with relative importance: $a > b > c > d > e$. For convenience, we set the initial weightings according to a binary interpretation of bit-patterns: $\alpha = 2^4$, $\beta = 2^3$, $\gamma = 2^2$, $\delta = 2^1$, and $\epsilon = 2^0$.

3 IMPLEMENTATION OF ANNEALING

Simulated annealing uses the principle of *local search*; starting from the initial configuration it iteratively performs small random changes to the configuration, in an attempt to minimize the objective function.

The optimization algorithm had three subroutines (①, ②, ③), one to optimize each set of parameters (see Section 2): the weightings were optimized in subroutine ①, the arrangement of bit-patterns into sets was optimized in subroutine ②, and the bit-pattern set to amino acid assignments were optimized in subroutine ③. While optimizing each of the three sets of parameters, the other two parameter set assignments were frozen. Subroutine ① involved optimizing five continuous variables using a one-dimensional gradient method. In contrast, subroutines ② and ③ involved combinatorial problems which were optimized by simulated annealing.

Each subroutine was called in turn, first subroutine ③, then subroutine ②, then subroutine ①, optimizing and repeating the ③–②–① cycle until self-consistency. This *multiple annealing* strategy, where the final solution from one optimization attempt was the starting configuration for the following attempt, is useful if there are many local minima near the global minimum (Pham and Karaboga, 2000).

Code to implement this algorithm was written in C and was partly based on published code (Press *et al.*, 1992).

3.0.1 Changing weighting factors. Subroutine ① was used to search for the optimal values of weightings α, β, γ, δ, and ϵ. Each weighting was optimized using a one-dimensional gradient method. First α was optimized, then β, γ, δ, ϵ, then α, β, ..., and so on. A new configuration corresponded to a new value for the weighting being optimized. For example, when optimizing α, the initial weighting was $\alpha = 16$. A new one could be $\alpha = 16 + x$. If this change would cause the objective function (E) to decrease, it was accepted and $\alpha = 16 + 2x$ was assessed.

Table 1. The bit-pattern set to amino acid mapping used as the initial configuration. The thirty-two bit-patterns were divided into twenty sets; one bit-pattern was then selected from each set to find the twenty patterns that minimized the objective function

P	P	**P**	P	**A**	A	G	**G**	**Q**	**N**	**E**	**D**	T	**T**	S	**S**	C	**C**	C	C	**V**	**I**	**M**	**L**	**F**	**Y**	**W**	**H**	**K**	K	**R**	R	
0	0	**0**	0	**0**	0	0	**0**	**0**	**0**	**0**	**0**	0	**0**	0	**0**	1	**1**	1	1	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	1	**1**	1	a
0	0	**0**	0	**0**	0	0	**0**	**1**	**1**	**1**	**1**	1	**1**	1	**1**	0	**0**	0	0	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**1**	1	**1**	1	b
0	0	**0**	0	**1**	1	1	**1**	**0**	**0**	**0**	**0**	1	**1**	1	**1**	0	**0**	0	0	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**1**	1	**1**	1	c
0	0	**1**	1	**0**	0	1	**1**	**0**	**0**	**1**	**1**	0	**0**	1	**1**	0	**0**	1	1	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**1**	**0**	0	**1**	1	d
0	1	**0**	1	**0**	1	0	**1**	**0**	**1**	**0**	**1**	0	**1**	0	**1**	0	**1**	0	1	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	1	**0**	1	e

P	**A**	**G**	**Q**	**N**	**E**	**D**	**T**	**S**	**C**	**V**	**I**	**M**	**L**	**F**	**Y**	**W**	**H**	**K**	**R**	
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	a
0	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	b
0	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**1**	c
1	**0**	**1**	**0**	**0**	**1**	**1**	**0**	**1**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**1**	**0**	**1**	d
0	**0**	**1**	**0**	**1**	**0**	**1**	**1**	**1**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**0**	e

However, if the change meant that E would increase, $\alpha = 16 + x$ was rejected and $\alpha = 16 - x$ was assessed. When the situation was reached where either adding or subtracting x would cause E to increase, E could not be minimized further by optimizing α, so then β was optimized, then $\gamma, \ldots,$ and so on.

3.0.2 Changing bit-pattern sets. There are ${}^{32}C_{20}$, or approximately 2.3×10^8, ways that twenty patterns can be chosen from the thirty-two possible five-bit patterns. The aim of subroutine ② was to search for the optimal choice of twenty bit-patterns. A new configuration corresponded to a different arrangement of the thirty-two patterns in twenty sets; one bit-pattern was then selected from each set to find the combination of twenty patterns that minimized E (Table 1). That is, for a particular arrangement of the thirty-two patterns into twenty sets, the objective function value was taken to be the minimum found when the objective function was evaluated for every possible selection of twenty patterns (one from each set). Bit-pattern sets were constrained to contain patterns that were adjacent in the initial configuration (Table 1). For example, in the initial configuration the first four bit-patterns

$$\begin{matrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{matrix}$$

formed one set, while just the first three

$$\begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{matrix}$$

may be in the same set in a new configuration.

3.0.3 Changing pattern set to residue map. There are 20!, or approximately 2.4×10^{18}, ways that twenty bit-pattern sets can be assigned to twenty amino acids. Subroutine ③ was used to search for the best way. A new configuration corresponded to a new bit-pattern set to amino acid mapping, obtained by swapping the bit-patterns assigned to a pair of amino acids, for one or more pairs of amino acids. For example, in the initial configuration amino acids M and V were represented by (10110) and (10100), respectively, while in a new configuration these could be swapped.

3.1 Objective function

To derive the objective function, a matrix A of dissimilarities between bit-patterns was calculated, using the weighted XOR distance between patterns as a measure of their dissimilarity. A matrix D of mutational distances between residues was calculated, estimating mutational distance from the BLOSUM45 mutation matrix (Henikoff and Henikoff, 1992). The purpose of the optimization algorithm was to maximize the correlation of the mutational distances between amino acids, estimated by D, and the corresponding bit-patterns' dissimilarities, calculated as A. The objective function (E) was the square of the Euclidean distance between matrices A and D; the optimization algorithm minimized this A–D distance.

The BLOSUM45 matrix was chosen as the mutation matrix for the following reason. The quality of the solution from simulated annealing depends on having a starting configuration near the global optimum. Hence, we wanted to force our initial configuration, which was chosen such that chemically similar amino acids were given similar bit-patterns (Table 1), to be near the global optimum. Chemical differences between amino acids are reflected in mutation matrices built from highly divergent sequences (Tomii and Kanehisa, 1996). The BLOSUM45 mutation matrix, the construction of which involves weighting divergent sequences much more than similar sequences, was an appropriate matrix (Henikoff and Henikoff, 1992).

3.1.1 Distance matrix D. The distance matrix D was derived from the BLOSUM45 mutation matrix using the inter-row distance method (Taylor and Jones, 1993). This method treats the rows of BLOSUM45, which represent amino acids, as vectors defining points in a twenty-dimensional space. To define D the Euclidean distance between the points was calculated:

$$D_{ij} = \sqrt{\sum_{k=1}^{20}(\text{BLOSUM45}_{ik} - \text{BLOSUM45}_{jk})^2}.$$

3.1.2 Dissimilarity matrix A. To derive matrix A, the distance between two bit-patterns was calculated as the weighted XOR of the bits. For example, if the weightings were $\alpha = 16$, $\beta = 8$, $\gamma = 4$, $\delta = 2$, and $\epsilon = 1$, then the distance between (10100) and (00110) would be:

$$\begin{aligned}\alpha(a_1 \oplus a_2) + \beta(b_1 \oplus b_2) + \gamma(c_1 \oplus c_2) + \delta(d_1 \oplus d_2) \\ + \epsilon(e_1 \oplus e_2) \\ = (0 \times 16) + (1 \times 8) + (1 \times 4) + (0 \times 1) = 13.\end{aligned}$$

We calculated A_{ij} to be the average of the weighted XORs of the bit-patterns in the set representing amino acid i and the bit-patterns in the set representing amino acid j. To save time, the objective function was evaluated by calculating A in this way in subroutines ① and ③, while in subroutine ② the objective function was evaluated by the more thorough and time-consuming method of taking the minimum objective function value for all possible choices of one bit-pattern from each set (Section 3.0.2). This was feasible because less configurations were assessed in subroutine ② than ① or ③.

3.1.3 Calculating distance between A and D. When comparing matrices the following steps must be taken (Risler *et al.*, 1988; Zintzaras and Kowald, 1999):

(1) The matrices A and D had to have comparable sizes, so were scaled to have the same sum over all elements. We defined D_{sum} as:

$$D_{\text{sum}} = \sum_{i=1}^{20}\sum_{j=1}^{20} D_{ij}$$

and A_{sum} as:

$$A_{\text{sum}} = \sum_{i=1}^{20}\sum_{j=1}^{20} A_{ij}.$$

Matrix A was then scaled to create A':

$$A'_{ij} = \frac{D_{\text{sum}}}{A_{\text{sum}}} A_{ij}.$$

(2) Then A' and D were expressed relative to their centroids to remove translation effects from the distance calculated. If the mean of column j in matrix A' is:

$$\bar{A'}_j = 1/20 \sum_{i=1}^{20} A'_{ij}$$

then the centroid of A' has coordinates ($\bar{A'}_1$, $\bar{A'}_2, \ldots, \bar{A'}_{20}$). The centroid of D is similarly defined. If M is the Euclidean distance between A' and D, then M^2 is:

$$M^2 = \sum_{i=1}^{20}\left\{\sum_{j=1}^{20}(A'_{ij} - D_{ij})^2\right\}$$

which can be rewritten as:

$$\begin{aligned}M^2 = \sum_{i=1}^{20}\sum_{j=1}^{20}\{(A'_{ij} - \bar{A'}_j) - (D_{ij} - \bar{D}_j)\}^2 \\ + 20\sum_{j=1}^{20}(\bar{A'}_j - \bar{D}_j)^2.\end{aligned}$$

That is:

$$M^2 = M_0^2 + 20\sum_{j=1}^{20}(\bar{A'}_j - \bar{D}_j)^2$$

where M_0^2 is the residual distance between A' and D after translation so that their centroids are at the origin, and $\sum_{j=1}^{20}(\bar{A'}_j - \bar{D}_j)^2$ is the square of the Euclidean distance between their centroids.

The objective function (E) was defined as $E = M_0^2$, where M_0^2 was the square of the Euclidean distance between A' and D after they had been expressed relative to their centroids.

3.2 The annealing schedule

In this preliminary study, the annealing schedule employed was determined heuristically, by examining the performance of each subroutine.

3.2.1 Optimizing weightings. In subroutine ① weightings α, β, γ, δ, and ϵ were optimized in turn, and until self-consistency. For each step (for example, a step optimizing δ), new configurations were tested until convergence was reached, subject to at least 420 new configurations having been accepted. A new configuration corresponded to $\delta_{\text{new}} = \delta_{\text{old}} + x$ or $\delta_{\text{new}} = \delta_{\text{old}} - x$.

3.2.2 Optimizing bit-pattern sets. Press *et al.* (1992) advise *choosing a starting value for the parameter T which is considerably larger than the largest* ΔE *normally encountered*, so in subroutine ② the initial T value was set to 80. Each time T was changed, it was reduced by 1%. Before each change in T occurred, T was kept constant while 100 new configurations were assessed. Each new configuration corresponded to a different arrangement of the thirty-two five-bit patterns in twenty sets. If the number of new configurations that had been accepted surpassed 70 for a particular value of T, T was reduced. The best solution so far was saved and was adopted as the starting configuration each time that T was changed. In this preliminary study we assumed that the initial bit-pattern sets were reasonable, so bit-pattern sets were constrained to contain patterns that were adjacent in the initial configuration. However, future calculations could constrain sets to have bit-patterns with the same most highly weighted bits. In practice, we found that the objective function was not very sensitive to changes in the arrangement of bit-patterns into sets, so it was only necessary to change T twice.

3.2.3 Optimizing pattern set to residue map. In subroutine ③ the value of T was changed 1000 times. The initial T, T_0, was 40, near the largest decrease in E found. Each time T was changed, it was reduced by 10%. Before each change in T occurred, T was kept constant while 100 000 new configurations were assessed. A new configuration corresponded to a new bit-pattern set to amino acid mapping. If the number of new configurations that had been accepted surpassed 70 000 for a particular value of T, T was reduced. To avoid being trapped in local minimum, a *kickstart* was programmed after every 500 accepted configurations: T was set to 1000 until ten more configuration changes had been accepted. The pre-kickstart configuration was stored and was readopted if a better solution was not reached by the time fifteen more configuration changes had been accepted. Subroutine ③ ran until T had been changed 1000 times; or until 100 000 configurations had been assessed without accepting even one, for a particular value of T. On average T was changed 29 times. The best solution so far during subroutine ③ was saved and used as the starting configuration for subroutine ②.

After 70 000 configurations had been assessed and T was due to be decreased for the first time, E was still oscillating. If T is changed before assessing enough configurations, simulated annealing can converge to a local rather than the global minimum. To check whether the oscillations were between similar or different solutions, every 150th configuration accepted while T was equal to T_0 was sampled, taking nine samples. For each of these nine samples the weightings and bit-pattern sets were the same as in the initial configuration (Table 1), while the bit-pattern set to amino acid assignments were different. The distance between these configurations was calculated as that between the corresponding matrices A (the distance M_0^2), after scaling the matrices relative to BLOSUM45. Using the program `neighbor` (Felsenstein, 1993), a neighbour-joining tree was drawn to compare the samples to 200 random configurations. Compared to the random configurations, the samples were clustered.

The nine samples were then compared to four configurations (*C1–C4*) generated from the initial configuration. Configuration *C1* was generated by swapping the bit-pattern sets for A and P, while *C2* was generated by swapping those for P and G; both changes involved similar residues. In contrast, configuration *C3* was generated by swapping the bit-pattern sets for P and R, while *C4* was generated by swapping those for A and C; here both changes involved dissimilar residues. The nine configurations sampled during annealing were as similar to each other as *C1*, *C2* and the initial configuration were to each other. Thus subroutine ③ was cycling between similar solutions.

To find out whether the space searched by subroutine ③ had hundreds of local minima or just a few, random initial configurations were generated, subroutine ③ was run, and the solutions compared. If there were a unique global minimum, random startpoints would converge to the same solution. A *quenching* version of subroutine ③ was also run. The quenching version had $T = 0$, so it rejected all configurations having higher E, thereby increasing the likelihood of being trapped in a local minimum. A tree of the initial and final configurations was drawn using the program `neighbor` (Felsenstein, 1993; Figure 1). For both annealing and quenching, random startpoints converged to closely related solutions. Hence, there was either one globally optimum bit-pattern set to amino acid mapping with many nearby local optima, or a cluster of several equally good global optima.

4 RESULTS AND DISCUSSION

4.1 Significance of the bit weightings

When subroutine ① was run until convergence, keeping the arrangement of bit-patterns in sets and the bit-pattern set to amino acid mapping as in the initial configuration, the solution found was $\alpha \approx 7.64$, $\beta = 6.16$, $\gamma = 5.07$, $\delta = 5.15$, and $\epsilon = 8.71$. However this solution was not the best attainable, since the arrangement of bit-patterns in sets and the bit-pattern set to amino acid mapping had not been optimized.

Expressed as a fraction of $(\alpha + \beta + \gamma + \delta + \epsilon)$, $(\epsilon + \alpha)$ was 50%, and $(\epsilon + \alpha + \beta)$ was 69%. Thus three bits may be enough to capture the basic interrelationships of the amino acids, as expressed in the BLOSUM45 matrix. This agrees with Taylor and Jones' (1993) conclusion

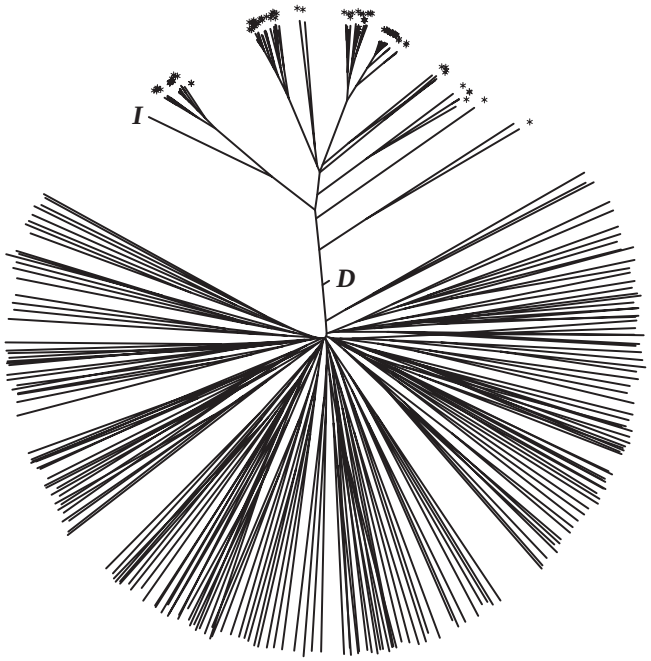

Fig. 1. A neighbour-joining tree of 195 random initial configurations used for simulated annealing or quenching in subroutine ③ (unmarked branches), the 100 different solutions reached after running subroutine ③ (branches marked with asterisks), the BLOSUM45-derived distance matrix (*D*), and the initial configuration (*I*). The solutions, *D*, and *I* are clustered at the top.

that three dimensions at the least are needed to explain mutational exchanges. Tomii and Kanehisa (1996) found that the main trends in the BLOSUM45 matrix are largely explained by two orthogonal factors; the third most important factor has a small effect. If each amino acid was to be represented by three bits, the three bits with the highest weightings would be taken: $\boxed{eab}$ in this case.

Buttimore and Mac Dónaill (1996) assigned bits an order of importance $a > b > c > d > e$, and gave residues of similar hydrophobicity the same *a* and of similar volume the same *b* (Table 1), assuming that hydrophobicity and volume are the major properties influencing amino acid substitution during evolution. This assumption seems correct, because after optimization the weightings were ordered in size: $\epsilon > \alpha > \beta > \delta > \gamma$, where α and β are the weightings for bits *a* and *b*, respectively. This indicated that hydrophobicity and volume are two of the physicochemical properties reflected most in BLOSUM45, as has been shown (Tomii and Kanehisa, 1996).

What does the result $\epsilon > \alpha > \beta$ mean? We wanted that the weighted XOR distances between the bit-patterns assigned to, for example, F, Y and W, would be proportional to the amino acids' dissimilarity according to BLOSUM45. For this to be true, the bits that distinguish F, Y, and W, which are bits *d* and *e*, must carry a high weight; a large δ or ϵ is necessary. This is because if bits *d* and *e* were ignored, and amino acids were partitioned according to just bits *a* and *b*, then V, I, M and L would be indistinguishable, as would F, Y, and W (Table 1). However, the distances *within* the V–I–M–L and F–Y–W clusters in BLOSUM45 are nearly as large as the distance *between* the clusters. For this relationship to be captured, the weighting ϵ on bit *e* had to be large. Clearly, the significance of bit *e* was underestimated in the initial guess.

4.2 Fully optimized mapping

The entire optimization algorithm comprising subroutines ①, ②, and ③ was run until convergence. Simulated annealing does not guarantee to find the optimum solution, so running the optimization algorithm for longer could have found a marginally better solution. However, for preliminary exploration of the envisaged filter algorithm a near-optimal solution is enough. The time of convergence of the simulated annealing algorithm was at most one day when run on a LINUX platform with Pentium II processors (400 MHz). Therefore, bit pattern representations corresponding to different substitution matrices can be calculated relatively quickly (a calculation that only needs to be done once for each substitution matrix).

This best solution (Table 2a) had weights $\alpha = 10.30$, $\beta = 6.11$, $\gamma = 5.83$, $\delta = 7.73$, and $\epsilon = 7.17$. The weightings were ordered $\alpha > \delta > \epsilon > \beta > \gamma$ in this solution. Unlike in the initial solution, the most highly weighted bits did not clearly partition amino acids according to physicochemical properties, and so α, β, γ, δ and ϵ could not be interpreted as chemical weightings in the final solution.

Interestingly, $(0.5 \times \alpha) \approx \beta \approx \gamma \approx \delta \approx \epsilon$. In other words, the distance between two five-bit patterns

$$\boxed{b_4 b_3 b_2 b_1 b_0} \text{ and } \boxed{b'_4 b'_3 b'_2 b'_1 b'_0}$$

can be crudely approximated as the weighted XOR distance between them calculated by setting all the weightings to one, that is, as the *Hamming distance* between them. When the weightings were optimized α was roughly twice as big as the other weightings; hence, the distance between the five-bit patterns is better approximated by the Hamming distance between the six-bit patterns

$$\boxed{b_4 b_4 b_3 b_2 b_1 b_0} \text{ and } \boxed{b'_4 b'_4 b'_3 b'_2 b'_1 b'_0}$$

so that the bit weighted by α, namely b_4, is counted twice. Using this Hamming distance as a scoring function would allow us to make use of integer arithmetic, which is much faster than floating-point arithmetic on typical computers.

When similarity between amino acids was estimated from BLOSUM62, the default matrix in the commonly

Table 2. The final solutions found when (a) the BLOSUM45 matrix and (b) the BLOSUM62 matrix were used to estimate similarity between amino acids

(a)

I	Y	K	M	W	L	R	F	H	T	E	D	Q	N	P	A	G	C	V	S	
0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	a
0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	1	1	1	1	1	b
0	0	0	1	0	0	1	1	0	0	1	1	1	1	0	0	0	1	1	1	c
0	1	1	0	0	0	0	1	0	1	0	0	1	1	0	0	1	0	0	1	d
0	0	1	0	0	1	0	1	0	0	0	1	0	1	0	1	0	0	1	0	e

(b)

K	Y	R	S	L	C	I	F	H	T	N	D	E	Q	P	A	G	M	W	V	
0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	a
0	0	0	1	1	1	1	1	0	0	0	0	0	0	1	1	1	1	1	1	b
0	0	1	0	0	0	0	1	0	0	0	0	1	1	0	0	0	0	1	1	c
0	1	0	0	0	1	1	0	0	0	1	1	1	1	0	0	1	1	0	1	d
1	0	0	0	1	0	1	1	0	1	0	1	0	1	0	1	0	1	0	1	e

used BLAST algorithm (Altschul *et al.*, 1997), the weightings were different from the BLOSUM45 result ($\alpha = 11.28$, $\beta = 7.05$, $\gamma = 6.47$, $\delta = 5.74$ and $\epsilon = 6.45$), as was the best bit-pattern to amino acid mapping found (Table 2b). The BLOSUM45 matrix was chosen so that the initial configuration would be close to the global optimum (see Section 3.1); if BLOSUM62 had been used, it would have been further away.

4.3 Application: comparing porins

To illustrate the behaviour of the envisaged filter algorithm, two porins, the bacteriophage *PA2* LC precursor (SWISSPROT P07238) and the *Escherichia coli* PhoE precursor (SWISSPROT P02932), were compared to each other using the Smith–Waterman algorithm (1981). Three different alignments were calculated, each with a different scoring function: with the BLOSUM45 matrix; with the matrix $A_{5\text{bit}}$ calculated from the best five-bit representation of amino acids found; and with the matrix $A_{3\text{bit}}$ calculated from the best three-bit representation of amino acids found, using the three most highly weighted bits *a*, *d*, and *e* (Figure 2). $A_{5\text{bit}}$ and $A_{3\text{bit}}$ were calculated as in Section 3.1.2. The three alignments were nearly the same, except that the positions where gaps were placed differed by two or three amino acids between alignments. This confirms the potential of a three-bit per amino acid filtration approach. If bits *a*, *b*, and *c* were used instead of *a*, *d*, and *e*, the alignments differed only slightly. This suggested that bits *b*, *c*, *d* and *e* are roughly equally important, and so the idea of a Hamming distance scoring function is not infeasible as long as bit *a* is counted twice.

4.4 Conclusion

This paper describes how we searched for the optimal bit-pattern to amino acid mapping, or equivalently for the optimal weighted XOR scoring function. We considered this to be an important step in developing a filter algorithm that would represent amino acids with bit-patterns, because the sensitivity and selectivity of any filter algorithm depends largely on the accuracy of the function it employs for scoring substitutions (Johnson and Overington, 1993; Pearson, 1995; Vogt *et al.*, 1995).

The best way to implement the filter algorithm will be the focus of future work; for example, we need to work out the most efficient way to preprocess amino acid sequences into bit patterns on-the-fly, to introduce insertions and deletions during the Smith–Waterman algorithm when several residues are stored in one machine word, and to estimate gap penalties. We also plan to compare the filtration strategy, using bit-patterns corresponding to different substitution matrices, with other filter algorithms in terms of its sensitivity and selectivity for detecting homologues at different evolutionary distances, and its speed.

ACKNOWLEDGEMENTS

The research of the first author was partly supported by the PRTL 1999 Scheme of the Higher Education Authority (HEA) of Ireland. The authors would like to thank Dr K.H.Wolfe and Dr A.T.Lloyd (Department of Genetics, University of Dublin, Trinity College) for very helpful discussions. They also wish to thank the referees for their useful remarks and interesting suggestions.

REFERENCES

Altschul,S.F., Madden,T.L., Schäffer,A.A., Zhang,J., Zhang,Z., Miller,W. and Lipman,D.J. (1997) Gapped BLAST and PSI-BLAST: a new generation ofprotein database search programs. *Nucleic Acids Res.*, **25**, 3389–3402.

Blaisdell,B.E. (1989) Effectiveness of measures requiring and not requiring prior sequence alignment for estimating the dissimilarity of natural sequences. *J. Mol. Evol.*, **29**, 526–537.

Buttimore,N.H. and Mac Dónaill,D.A. (1996) The exploitation of assembly language instructions in biological text manipulation: II. Amino acid sequences. *Comput. Math. Appl.*, **32**, 39–45.

Dumas,J.P. and Ninio,J. (1982) Efficient algorithms for folding and

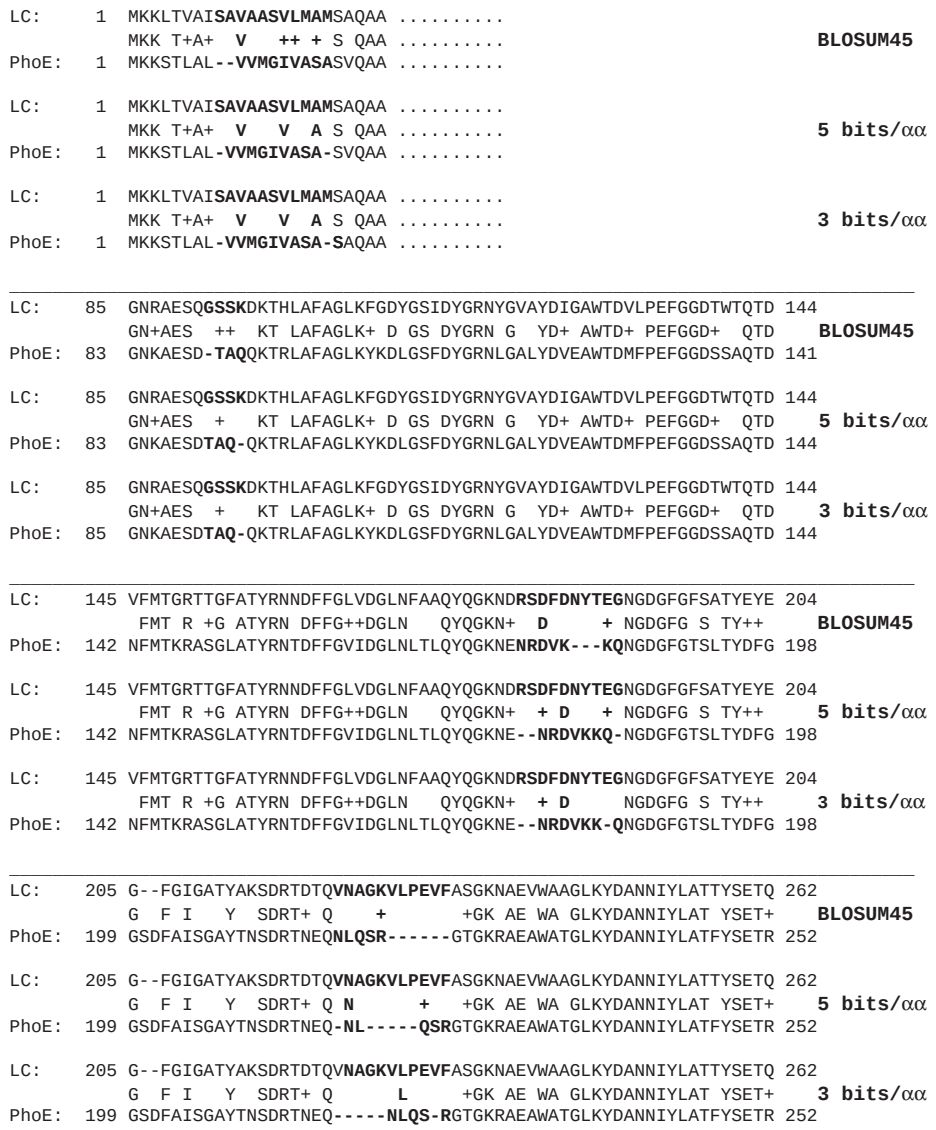

Fig. 2. Three alignments of the N-termini of bacteriophage *PA2* LC precursor and *E.coli* PhoE precursor. These were calculated using the BLOSUM45 scoring matrix, using a five-bit representation of amino acids, and using a three-bit representation of amino acids, respectively. The Smith–Waterman algorithm (1981) was used to calculate the alignments, using a gap opening penalty of −14 and a gap extension penalty of −2. Differences between the three alignments are highlighted in bold. Alignments are not shown for residues 26–84 of the LC precursor (24–82 of the PhoE precursor); the three alignments are identical in this region. The LC precursor is 366 amino acids long and the PhoE precursor is 352 amino acids long.

comparing nucleic acid sequences. *Nucleic Acids Res.*, **10**, 197–206.

Felsenstein,J. (1993) PHYLIP (Phylogeny Inference Package). Department of Genetics, University of Washington, Seattle, version 3.53c.

Green,P. (1996) SWAT: an efficient implementation of theSmith–Waterman or Needleman–Wunsch algorithms. http://bozeman.mbt.washington.edu/phrap.docs/swat.html.

van Heel,M. (1991) A new family of powerful multivariate statistical sequence analysis techniques. *J. Mol. Biol.*, **220**, 877–887.

Henikoff,S. and Henikoff,J.G. (1992) Amino acid substitution matrices from protein blocks. *Proc. Natl Acad. Sci. USA*, **89**, 10 915–10 919.

James,P., Quadroni,M., Carafoli,E. and Gonnet,G. (1993) Protein identification by mass profile fingerprinting. *Biochem. Biophys. Res. Commun.*, **195**, 58–64.

Johnson,M.S. and Overington,J.P. (1993) A structural basis for sequence comparisons—an evaluation of scoring methodologies.

J. Mol. Biol., **233**, 716–738.
Kirkpatrick,S., Gelatt,C.D. and Vecchi,M.P. (1983) Optimization by simulated annealing. *Science*, **220**, 671–680.
Metropolis,N., Rosenbluth,A., Rosenbluth,M., Teller,A. and Teller,E. (1953) Equations of state calculations by fast computing machines. *J. Chem. Phys.*, **21**, 1087–1091.
Myers,E.W. (1995) Guest editor's foreword. *Algorithmica*, **13**, 1–6.
Myers,G. (1999) A fast bit-vector algorithm for approximate string matching based on dynamic programming. *J. ACM*, **46**, 395–415.
Pearson,W.R. (1995) Comparison of methods for searching protein sequence databases. *Protein Sci.*, **4**, 1145–1160.
Pearson,W.R. and Lipman,D.J. (1988) Improved tools for biological sequence comparison. *Proc. Natl Acad. Sci. USA*, **85**, 2444–2448.
Petrilli,P. (1993) Classification of protein sequences by their dipeptide composition. *Comput. Appl. Biosci.*, **9**, 205–209.
Pevzner,P.A. and Waterman,M.S. (1995) Multiple filtration and approximate pattern matching. *Algorithmica*, **13**, 135–154.
Pham,D.T. and Karaboga,D. (2000) *Intelligent Optimisation Techniques: Genetic Algorithms, Tabu Search, Simulated Annealing and Neural Networks*. Springer, London.
Press,W.H., Teukolsky,S.A., Vetterling,W.T. and Flannery,B.P. (1992) *Numerical Recipes in C: The Art of Scientific Computing*, 2nd edn, Cambridge University Press, Cambridge, pp. 446–451.
Risler,J.L., Delorme,M.O., Delacroix,H. and Henaut,A. (1988) Amino acidsubstitutions in structurally related proteins—a patternrecognition approach. *J. Mol. Biol.*, **204**, 1019–1029.
Sibbald,P.R., Sommerfeldt,H. and Argos,P. (1991) Identification of proteins in sequence databases from amino acid composition data. *Anal. Biochem.*, **198**, 330–333.
Smith,T.F. and Waterman,M.S. (1981) Identification of common molecular subsequences. *J. Mol. Biol.*, **147**, 195–197.
Taylor,W.R. and Jones,D.T. (1993) Deriving an amino acid distance matrix. *J. Theor. Biol*, **164**, 65–83.
Tiourine,S., Hurkens,C. and Lenstra,J.K. (1995) An overview of algorithmic approaches to frequency assignment problems. *Technical Report*, CALMA (Combinatorial Algorithms for Military Applications), ftp://ftp.win.tue.nl/pub/techreports/CALMA/overview.ps.
Tomii,K. and Kanehisa,M. (1996) Analysis of amino acid indices and mutation matrices for sequence comparison and structure prediction of proteins. *Protein Eng.*, **9**, 27–36.
Vogt,G., Etzold,T. and Argos,P. (1995) An assessment of amino acid exchange matrices in aligning protein sequences: the twilight zone revisited. *J. Mol. Biol.*, **249**, 816–831.
Wozniak,A. (1997) Using video-oriented instructions to speed up sequence comparison. *Comput. Appl. Biosci.*, **13**, 145–150.
Zintzaras,E. and Kowald,A. (1999) A comparison of amino acid distance measures using procrustes analysis. *Comput. Biol. Med.*, **29**, 283–288.