Identifying Periodic Occurrences of a Template with Applications to Protein Structure *

Vincent A. Fischetti † *Gad M. Landau* ‡§ *Jeanette P. Schmidt* ‡¶
Peter H. Sellers †

Abstract

Consider a template P of size m in which each character matches many different characters with various degrees of perfection. Given a text T of size n, we present a simple and practical algorithm that finds the substring of T, which best matches some substring of P^n (P^n is the concatenation of an arbitrary number of copies of P). The algorithm produces the matched pair and their alignment in $O(mn)$ time.

Key Words: Computational Biology, Design of Algorithms, String Matching.

1 Introduction

Similarity among strings is a widely studied problem in computer science and other fields like Biology and Speech Recognition, see [SK-83] for details.

The most obvious way to measure the similarity of two substrings $a_r \ldots a_s$ and $b_k \ldots b_\ell$ is to find if they are equal (*exact similarity*). In many application, however, this measure is too strict and a more general measure is needed. *Approximate similarity* is a measure that is general enough for most applications. It defines the best alignment of two substrings with respect to a given scoring function. An alignment matches some characters and deletes others from both substrings. The application generally provides the scoring function, and thereby influences the alignment. In this paper we assume that the scoring is provided by two functions: a *gap function* that assigns penalties for gaps, and a match function that assigns scores to the matches. The *penalty matrix*, which is used as the match function, assigns a score to the match of each possible pair of characters. This score specifies how well the two characters "match". The penalty matrices considered here assign positive values to "favorable matches" and negative values to "unfavorable matches".

Definition: An alignment between the substrings $a_r \ldots a_s$ and $b_k \ldots b_\ell$ is specified by two strictly increasing sequences of indices ($r \leq i_1, i_2, \ldots i_w \leq s$) and ($k \leq j_1, j_2, \ldots j_w \leq \ell$), with the interpretation that for $h = 1 \cdots w$, a_{i_h} is matched to b_{j_h}, and all other characters in the substrings are unmatched. The score for matching the pair (a_{i_h}, b_{j_h}) is given by the penalty matrix M. The (always negative) score for leaving the text characters between a_{i_h} and $a_{i_{h+1}}$ and the pattern characters

*A preliminary version of this paper appeared in the Proceedings of the Third Combinatorial Pattern Matching Conference.

†Rockefeller University, 1230 York Avenue, New York, NY 10021.

‡Dept. of Computer Science, Polytechnic University, 333 Jay Street, Brooklyn, NY 11201.

§email: landau@pucs2.poly.edu, ((718) 260-3154). Partially supported by NSF grant CCR-8908286.

¶email: jps@pucs4.poly.edu, ((718) 260-3502). Partially supported by NSF grant CCR-9110255 and the New York State Science and Technology Foundation Center for Advanced Technology.

between b_{j_h} and $b_{j_{h+1}}$ unmatched is given by a gap function G with four parameters: $G(i_1, i_2, j_1, j_2)$ is the score for simultaneously deleting characters a_i, $i \in (i_1, i_2]$, from substring a and characters b_j, $j \in (j_1, j_2]$, from substring b. The score for the alignment is given by the following expression, where $i_{w+1} = s+1$, $j_{w+1} = \ell+1$, $i_0 = r-1$, $j_0 = k-1$: $\sum_{h=1}^{w} M(a_{i_h}, b_{j_h}) + \sum_{h=0}^{w} G(i_h, i_{h+1}-1, j_h, j_{h+1}-1)$.

An important special case occurs when the gap penalties are additive, and there is a function g, for which $G(i_1, i_2, j_1, j_2) = \sum_{i_1 < i \leq i_2} g(a_i) + \sum_{j_1 < j \leq j_2} g(b_j)$.

Many criteria have been suggested in the literature to compare two strings A and B. On one hand there are algorithms that measure the similarity of two entire strings (see for example [NW-69, WF-74, H-75]). In other applications one is interested in finding two substrings, one in A and one in B whose similarity score is highest among all such pairs of substrings, (as in [S-74, SW-81, We-73]). A third problem, frequently referred to as the "string matching problem", is that of finding all substrings in A, whose similarity score with the entire string B is above a given threshold, (e.g., [BM-77, GP-90, GS-83, KMP-77, KR-87, LV-89]). Other interesting string related problems and algorithms were addressed in [AG-85, D-90, SK-83, Wa-89].

In the problem discussed here, we would like to detect whether good matches to the pattern P (or to a cyclic rotation of P) are periodically repeated in the text. In our case the characters of the pattern come from a different alphabet than those from the text. The only description of a matching pair of characters is hence given by the penalty matrix. We define $\bar{P} = P^n$, such that P^n is built by concatenating n copies of P. OUR GOAL CAN BE RESTATED AS FINDING THE BEST MATCH BETWEEN A SUBSTRING OF $\bar{P}$ AND A SUBSTRING OF THE TEXT. Notice that the substrings of $\bar{P}$ can be characterized by a suffix of P followed by i copies of P followed by a prefix of P. Our algorithm is particularly useful if m is much smaller than n, and is designed to find many repetitions of P. We show that the structure of P^n can be exploited and the problem reduced to essentially solving a simple dynamic programming problem of size $O(mn)$. If gap penalties are either additive or can be computed in each step of the dynamic programming problem in $O(1)$ time, our algorithm runs in $O(mn)$ time.

Given a regular expression Q, Myers and Miller, [MM-89], give an $O(n|Q|)$ algorithm for finding the substrings of a given text that approximately match the regular expression Q. The set of *substrings* of P^* *is not* a regular expression of size m, but the algorithm in [MM-89] can be modified to close this gap. Our algorithm, while solving a more specialized problem, is much simpler than theirs.

Section 2 contains a short description of the motivation of the problem, which comes from protein structure determination. The exact definition of the problem is given in Section 3. Section 4 describes a simple dynamic programming algorithm that solves the problem by computing $O(mn^2)$ values. Section 5 describes the efficient $O(mn)$ time algorithm.

2 Protein structure motivation

The secondary structure of proteins is critical for their proper function. For the majority of proteins, the secondary structure can not simply be resolved by examining the sequence of the amino acids that define the protein. While x-ray crystallography is the only reliable means available today by which to solve the structure of a protein molecule, it can not always be used. The *α-helical* structure is one of the most common structures because of its ability in stabilizing proteins through short regions of helix-helix packing, referred to as a coiled-coil interaction, ([CP-86], [FP-92]). The class of coiled-coil proteins is much more predictable as to their secondary and tertiary structure, by examining the amino acids that define the protein. The ability to form an α-helical coiled-coil structure is based on the presence of a seven-residue repeat pattern denoted by (a-b-c-d-e-f-g). The first and the fourth position, (a and d), are generally apolar or hydrophobic amino acids which, in the context of the α-helix, form an inclined stripe around the axis of the helix [CP-86]. Discontinuities in the heptad pattern, such as changes in frame have been seen in coiled-coil molecules. To date, the method used to determine the presence of a heptad pattern in protein molecules is by inspection.

In the above application the pattern consists of the seven positions (a-b-c-d-e-f-g), while the alphabet of the text consists of the usual alphabet that represents proteins: the 20 amino acids. A penalty matrix determines how well a given amino acid fits a given position in the pattern. A protein features a coiled-coil structure if it contains a good match to a substring of the pattern (a-b-c-d-e-f-g)^n. The method used to construct the penalty matrix is based on profile analysis, ([GLE-87], [LLE-91]). A large database of known coiled-coil proteins whose structure has been determined is used as a reference. In [CP-90] and [LD-91], such data for four known classes of coiled-coil proteins has been tabulated. In particular for each position j, (among the seven positions) the average frequency p_{ij} of the occurrence of amino acid i in position j has been recorded. This frequency is compared with the overall frequency P_i of amino acid i in GENBANK. When $p_{ij} > P_j$, amino acid j occurs more often in position j than we would expect if the placement were random. $\ln(p_{ij}/P_j)$ is therefore positive if amino acid i is a good fit for column j, and negative for a bad fit. To avoid values of $-\infty$, the corresponding value $M[i,j]$, used in our penalty matrix was $\ln((p_{ij}+\epsilon)/(P_i+\epsilon))$, (for some small $\epsilon > 0$). Gap penalties were chosen empirically.

Inspired by the above application, this paper addresses the problem of determining a repeat structure in a text for a pattern of arbitrary length and an arbitrarily specified penalty matrix. Based on the algorithm presented in this paper a program was developed which determines whether a protein exhibits the 7-residue periodicity typical for the coiled-coil structure [FP-92].

3 The problem

We are given:
(i) a text $T = t_1 \ldots t_n$ over an alphabet Σ_1;
(ii) a pattern $P = p_1 \ldots p_m$ over an alphabet Σ_2;
(iii) a penalty matrix M defined over $\Sigma_1 \times \Sigma_2$, with negative values for bad matches and positive values for good matches;
(iv) a gap function G, defined over all pairs (τ, ρ), where τ is a substring of T and ρ is a substring of any cyclic rotation of P.

Let $\bar{P} = P^n$, where P^n is built by concatenating n copies of P. The gap function $G(i_1, i_2, j_1, j_2)$ takes its parameters j_1 and j_2 from the indices of P, (as opposed to $\bar{P}$). $G(i_1, i_2, j_1, j_2)$, for $0 < i_1 \le i_2 \le n$ and $0 \le j_1, j_2 \le m$, is the penalty for simultaneously deleting the substring $t_{i_1+1} t_{i_1+2} \ldots t_{i_2}$ from the text and the substring of length $(j_2 - j_1) \pmod m$ in the periodic pattern which starts after character p_{j_1}. For convenience of notation we will denote the above substring simply by $p_{(j_1+1)} \ldots p_{j_2}$, (i.e. if $j_2 < j_1 < m$, $p_{(j_1+1)} \ldots p_{j_2}$ stands for $p_{j_1+1} \ldots p_m p_1 \ldots p_{j_2}$; if $j_2 < j_1 = m$ it stands for $p_1 \ldots p_{j_2}$ and if $j_2 = j_1$ it denotes the empty substring). Note that:
(1) the string $p_{(j_1+1)} \ldots p_{j_2}$ represents all substrings $\bar{p}_{(x_1+1)} \bar{p}_{(x_1+2)} \ldots \bar{p}_{x_2}$ of the same length in $\bar{P}$, for which $x_1 = j_1 \pmod m$ and $x_2 = j_2 \pmod m$;
(2) substrings of length m or more in $\bar{P}$ do not have a representative, since there could not possibly be a gain of "jumping" over an entire period of the pattern, and therefore the corresponding gap penalty has not been defined.

We also require G to obey the following properties:

$$G(i,i,j,j) = 0 \qquad (a)$$

$$\text{for } i_1 < i_2 \text{ or } j_1 \neq j_2 \qquad G(i_1,i_2,j_1,j_2) \leq 0 \qquad (b)$$

as well as the triangle inequality:

$$\forall i_1 \leq i \leq i_2, \text{ and } \forall j_1, j, j_2 \qquad |G(i_1,i_2,j_1,j_2)| \leq |G(i_1,i,j_1,j)| + |G(i,i_2,j,j_2)| \qquad (c)$$

Definition: An optimal alignment between $\bar{P}$ and T is an alignment of a *substring* of $\bar{P}$ and a *substring* of T that yields the maximum score. (The alignments of substrings and their score as a function of M and G was defined in the Introduction).

The aim of our algorithm is to produce an optimal alignment between T and $\bar{P}$.

This entails not simply finding the best alignment of two given substrings, but also choosing the best substrings. Note that the alignment of the *chosen* substrings will always start and end with a match that carries a positive score.

In general, in order to find the alignment with the highest score efficiently one needs to make some assumptions on the "civilized behavior" of the gap penalties, such as for example in [G-82, GG-89, MM-88]. In the present article we are not overly concerned how gap penalties are chosen, since the main contribution of the present work is a reduction from a large size dynamic programming problem to a much smaller scale dynamic programming program. Any assumptions on the gap penalties can be carried from one solution to the other.

4 A simple but inefficient algorithm

Dynamic programming is a well known technique that is frequently used to solve problems related to approximate matching ([GP-90, LV-89, NW-69, S-74, SK-83]). In this section we show how a dynamic programming problem of size $O(mn^2)$ can be used to solve our problem. Let $S_{\ell,j}$ be the set of all pairs of strings (x,y), for which x is a suffix of $t_1t_2\ldots t_\ell$ and y is a suffix of $\bar{p}_1\bar{p}_2\ldots\bar{p}_j$. We construct a $n \times nm$ matrix D whose entry $D[\ell,j]$ equals $max_{x,y\in S_{\ell,j}} s(x,y)$, where $s(x,y)$ is the score of the optimal alignment between x and y; if this turns out to be negative, then define $D[\ell,j]=0$. Assigning 0 to $D[\ell,j]$ allows an alignment to start with the match of character $t_{\ell+1}$ with character $\bar{p}_{j+1}$, as observed in [SW-81]. The maximum value in D can be traced back to obtain the desired alignment.

The following algorithm computes the matrix D, when G obeys properties (a), (b) and (c) from the previous section.

Notation: We denote by $\bar{G}$ the gap function defined on substrings of size less than m of $\bar{P}$: $\forall i_1 \leq i_2, j_1 \leq j_2,\ \bar{G}(i_1,i_2,j_1,j_2) = G(i_1,i_2,j_1 \pmod m, j_2 \pmod m)$.

for $i := 0$ to n do
  for $j := 0$ to mn do
    $D[i,j] := 0$
for $i := 1$ to n do
  for $j := 1$ to mn do
  $(*) D[i,j] := \max\{D[i-1,j-1] + M[t_i, p_{j \bmod m}];\ \max_{\substack{i_0 \leq i \\ j-m<j_0\leq j}} D[i_0,j_0] + \bar{G}(i_0,i,j_0,j)\}$

  ($D[i,j]$ is the maximum of up to $im+1$ numbers. These numbers are 0 (the current value of $D[i,j]$), the value obtained from its predecessors on its diagonal followed by a match, and the $im-1$ predecessors in the upper left rectangle of width m respectively followed by a gap.)

od

od

Remark: Gap functions are frequently quite simple functions and generally the maximum in $(*)$ can be computed much faster. As is well known, when the gap function G is additive, the maximum in $(*)$ taken in the above program can be computed in $O(1)$ time by considering only the three neighbors of $D[i,j]$, ($D[i-1,j-1]$, $D[i-1,j]$ and $D[i,j-1]$). If G^* is an additive gap penalty and for some constant γ, the gap function G is defined as $G(i,i_1,j,j_1) = \max\{G^*(i,i_1,j,j_1),\gamma\}$, (which is a very suitable gap function for our problem), $D[i,j]$ may also be computed in $O(1)$ time, by considering four previously computed values, (the three values mentioned before and the maximal score computed so far).

Example: Let $t_1 \dots t_9$ be *cabaabddc*, let P be *ab*, and assume for simplicity that all deletions and insertions of i characters carry a penalty of $-i$, and that $M(x,y) = 1$ if $x = y$ and -1 otherwise. The matrix D that corresponds to these strings is given below:

		0	1	2	3	4	5	6	7	8	···
			a	b	a	b	a	b	a	b	···
0		0	0	0	0	0	0	0	0	0	···
1	c	0	0	0	0	0	0	0	0	0	···
2	a	0	**1** $\searrow$	0	**1** $\searrow$	0	1	0	1	0	···
3	b	0	0	**2** $\searrow$	1	**2** $\searrow$	1	2	1	2	···
4	a	0	1	1	**3** $\rightarrow$	**2** $\searrow$	**3** $\rightarrow$	**2** $\searrow$	3	2	···
5	a	0	1	0	2	2	**3** $\searrow$	2	**3** $\searrow$	2	···
6	b	0	0	2	1	3	2	**4**	3	**4**	···
7	d	0	0	1	1	2	2	3	3	3	···
8	d	0	0	0	0	1	1	2	2	2	···
9	c	0	0	0	0	0	0	1	1	1	···

Notice that the two optimal alignments found correspond actually to the same alignment.

5 The new algorithm

The new algorithm computes a matrix $\hat{D}$ of size $O(nm)$. $\hat{D}[i,j]$, (for $j = 1 \dots m$), will hold the score of the best alignment of a substring of T that ends with t_i and a substring of $\bar{P}$ that ends with any $\bar{p}_k$ for which $k = j \pmod m$, (in which case $p_k = p_j$). The above example illustrates why it seems sufficient to compute and keep only m locations in each row. For any j, the values $D[i,j], D[i,j+m], D[i,j+2m] \dots$ form an increasing sequence of scores. This follows directly from the fact that any suffix of $\bar{P}_j = \bar{p}_1, \dots \bar{p}_j$ is also a suffix of $\bar{P}_{j+m}$ and therefore the set of alignments competing for the maximal score when aligning with $\bar{P}_j$ is included in the corresponding set for $\bar{P}_{j+m}$. Since the largest number of pattern characters one might delete per character of text is $m-1$, it is easy to see that the values in row i will therefore be periodic starting from entry $D[i, im]$, (at the latest). The matrix $\hat{D}$ will therefore consist of the values in the above mentioned periods and contain the maximum value of D. We are going to show that each entry in $\hat{D}$ can be computed as efficiently as an entry in D. The maximum value in $\hat{D}$ can be easily traced back to obtain the desired alignment.

We shall now describe the computation of $\hat{D}$.

In the dynamic programming computation of D, an entry $D[i,j] > 0$ is obtained from the score computed at some entry $D[r,s]$, for $r \le i$ and $s \le j$. As in the "trivial algorithm" $\hat{D}[0,j]$ and $\hat{D}[i,0]$ are initialized to 0 for all i,j. If $D[i,j]$ is in the periodic region, then $\hat{D}[i, j \bmod m]$ will be computed from the score $\hat{D}[i_0, j_0]$, with $i_0 = r$ and $j_0 = s \bmod m$. As a result a score $\hat{D}[i,j]$, may depend on a score $\hat{D}[i_0, j_0]$, with $j_0 > j$. When $i_0 < i$, $\hat{D}[i_0, j_0]$ is computed before $\hat{D}[i,j]$. However, if $i_0 = i$ this is not the case. To overcome this difficulty the computation of each of the rows of $\hat{D}$ will be done in two rounds. In the first round we compute each of the entries $\hat{D}[i,j]$, $(j > 0)$, using all previously computed values in $\hat{D}$, assuming for the moment that all the entries in row i are correct. In the second round we feed the value $\hat{D}[i,m]$ back to $\hat{D}[i,0]$, (so that this value is no longer necessarily zero), and correct those values $\hat{D}[i,j]$, which depend on a score $\hat{D}[i,j_0]$, for $j_0 > j$. We shall prove that all entries in $\hat{D}[i; 0 \dots m]$ will have been correctly computed by the end of this second round.

Assume that the submatrix $\hat{D}[1 \dots i-1; 1 \dots m]$ has been correctly computed. To compute row i we recall that the entry $\hat{D}[i,j]$ is the maximum score of an alignment of any suffix of $t_1 \dots t_i$ and any substring of $\bar{P}$ that ends with p_j. This alignment may either end with the match of p_j and t_i or with a gap. We distinguish between two type of gaps. In the first type, the gap is in the text, and possibly also in the pattern, i.e. the last match in the alignment is between p_k and t_ℓ, for $\ell < i$ and any k. In the second type, the gap is in the pattern only, i.e. the last match in the alignment is between p_k and t_i, for some $k \neq j$, followed by the deletion of $p_{k+1} \ldots p_j$.

The three cases listed above correspond to the following computations:

(a) If the alignment is obtained from an alignment of a suffix of $t_1 \ldots t_{i-1}$, and a substring of $\bar{P}$ that ends with p_{j-1}, followed by the match of t_i and p_j, then $\hat{D}[i,j] := \hat{D}[i-1,j-1] + M[t_i, p_j]$.

(b) If the alignment is obtained from an alignment of a suffix of $t_1 \ldots t_\ell$, $\ell < i$ and a substring of $\bar{P}$ for which t_ℓ is matched to p_k (for any k), the subsequent deletion of both $t_{\ell+1} \ldots t_i$ and $p_{k+1} \ldots p_j$, then $\hat{D}[i,j] := \hat{D}[\ell,k] + G(\ell,i,k,j)$. (If $k = j$ no characters from the pattern are deleted.)

(c) If the alignment is obtained from an alignment of a suffix of $t_1 \ldots t_i$, and a substring of $\bar{P}$ that ends with a match of t_i and p_k (for any $k \neq j$) and the deletion of the substring $p_{k+1} \ldots p_j$ in the pattern, then $\hat{D}[i,j] := \hat{D}[i,k] + G(i,i,k,j)$.

In the first round we compute the scores for alignments corresponding to cases (a) and (b) above, since theses scores depend only on previously correctly computed scores. In the second round we consider case (c). Suppose that $\hat{D}[i,j]$ gets a higher score in the second round than in the first round. It follows that $\hat{D}[i,j]$ was obtained from the score at $\hat{D}[i,k]$, followed by the deletion of $p_{k+1} \ldots p_j$. Since G obeys the triangle inequality, and $\hat{D}[i,j]$ could not get its best score directly by case (b), the alignment corresponding to $\hat{D}[i,k]$ must have ended with a match, and was therefore correctly computed in the first round. After the second round all entries will therefore be correct, and the time to compute an entry in $\hat{D}$ is at most twice the one to compute an entry in D.

By the previous claim the following algorithm computes the values of the matrix $\hat{D}$ correctly.

Notation: The symbol $\preceq$ denotes the lexicographic order or pairs, i.e. $(i,j) \preceq (i_1, j_1)$ if $(i < i_1)$ or $((i = i_1)$ and $(j \leq j_1))$.

for $i := 0$ to n do
  for $j := 0$ to m do
    $\hat{D}[i,j] := 0$
for $i := 1$ to n do
  for $j := 1$ to m do
    $\hat{D}[i,j] := \max\{\hat{D}[i-1,j-1] + M[t_i,p_j];\ \max_{(i_0,j_0) \preceq (i,j)} \hat{D}[i_0,j_0] + G(i_0,i,j_0,j)\}$
  od
  $\hat{D}[i,0] := \hat{D}[i,m]$
  for $j := 1$ to m do
    $\hat{D}[i,j] := \max\{\hat{D}[i,j];\ \max_{0 \leq j_0 < m} \hat{D}[i,j_0] + G(i,i,j_0,j)\}$
  od
od

Remark: As mentioned earlier when the gap function is additive, we need only consider three neighbors for each entry. Therefore, in round 1 $\hat{D}[i,j]$ is given by:
$\max\{0; \hat{D}[i-1,j-1] + M[t_i,p_j]; \hat{D}[i_0,j_0-1] + G(i_0,i_0,j_0-1,j_0); \hat{D}[i_0-1,j_0] + G(i_0-1,i_0,j_0,j_0);\}$
Since gaps can only reduce the score, the maximal score in $\hat{D}[i, 1 \ldots m]$, will always have been correctly computed in the first round. Moreover, because G is additive, all scores to the right of the maximum will also be correct. In particular $\hat{D}[i,m]$ (and therefore

$\hat{D}[i,0]$) will have the right score after the first round. In the second round we compute $\hat{D}[i,j] := \max\{\hat{D}[i,j]; \hat{D}[i,j-1] + G(i,i,j-1,j)\}$, and the remaining scores (if any) will be adjusted. The above argument also shows that, (when gap values are additive), as soon as a value computed in the first round equals a value computed in the second round, the second round can be terminated.

Complexity: We compute a matrix of size $n(m+1)$. If the full size dynamic programming problem could be solved in $O(gn^2m)$ time, where g accounts for the complexity of computing the gap function, then our reduced dynamic programming problem can be solved in $O(gnm)$ time. In particular for additive gap functions the complexity is $\Theta(nm)$.

Example Let $t_1 \ldots t_9$ be *cabaabddc* and let P be *ab*. The matrix $\hat{D}$ corresponding to this pattern matching problem is given below:

		0	*1*	*2*
		(b)	*a*	*b*
0		*0*	*0*	*0*
1	*c*	*0*	*0*	*0*
2	*a*	*0*	**1** $\searrow$	*0*
3	*b*	$\rightarrow$ **2** $\searrow$	*1*	**2** $\rightarrow$
4	*a*	$\rightarrow$ **2** $\searrow$	**3** $\rightarrow$	**2** $\rightarrow$
5	*a*	*2*	**3** $\searrow$	*2*
6	*b*	*4*	*3*	**4**
7	*d*	*3*	*3*	*3*
8	*d*	*2*	*2*	*2*
9	*c*	*1*	*1*	*1*

References

[AG-85] A. Apostolico and Z. Galil (editors), *Combinatorial Algorithms on Words,* NATO ASI Series, Series F: Computer and System Sciences, Vol. 12, Springer-Verlag, 1985.

[BM-77] R.S. Boyer and J.S. Moore, "A fast string searching algorithm," *CACM*, Vol. 20, 1977, pp. 762-772.

[CF-74] P.Y. Chou and G.D. Fasman, "Prediction of protein conformation," *Biochemistry,* Vol. 13, 1974, pp. 222-245.

[CP-86] C. Cohen, and D.A.D. Parry, "Alpha-helical coiled coils – a widespread motif in proteins," *T.I.B.S.,* Vol. 11, 1986, pp. 245-248.

[CP-90] J. F. Conway and D. A. D. Parry, "Structural features in the heptad substructure and longer range repeats of two-stranded *alpha*-fibrous proteins," *Int. J. Biol. Macromol.*, Vol. 4, 1990, pp. 328-333.

[D-90] R. F. Doolittle, (editor) (1990) *Methods in Enzymology,* Vol. 183: Molecular Evolution: Computer Analysis of Protein and Nucleic Acid Sequences.

[FP-92] V. A. Fischetti, V. Pancholi, P. Sellers, J. Schmidt, G. Landau, X. Xu, O. Schneewind, "streptococcal M protein: A common structural motif used by Gram-positive bacteria for biological active surface molecules" to appear *Molecular Recognition in Host-Parasite Interactions: Mechanisms in viral, bacterial and parasite infections.* Published by Plenum Publishing.

[G-82] O. Goton, "An improved algorithm for matching Biological sequences," *J. Molecular Biology,* Vol. 162, 1982, pp. 705-708.

[GG-89] Z. Galil and R. Giancarlo, "Speeding up dynamic programming with applications to molecular biology," *Theoretical Computer Science,* Vol. 64, 1989, pp. 107-118.

[GLE-87] M. Gribskov, A. D. McLachlan, and D. Eisenberg, "Profile analysis: detection of distantly related proteins," *Proc. Natl. Acad. Sci.,* Vol. 84, 1987, pp. 4355-4358.

[GOR-78] J. Garnier, D.J. Osguthorpe, and B. Robson, "Analysis of the accuracy and implications of simple methods for predicting the secondary structure of globular proteins," *J. Molecular Biology,* Vol. 120, 1978, pp. 97-120.

[GP-90] Z. Galil and K. Park, "An improved algorithm for approximate string matching," *SIAM J. Comp.,* Vol. 19, 1990, pp. 989-999.

[GS-83] Z. Galil and J. I. Seiferas, "Time-space optimal string matching," *J. Computer and Systems Sciences*, Vol. 26, 1983, pp. 280-294.

[H-75] D. Hirschberg, "A linear space algorithm for computing maximal common subsequences," *CACM,* Vol. 18, No 6, 1975, pp. 341-343.

[KMP-77] D.E. Knuth, J.H. Morris and V.R. Pratt, "Fast pattern matching in strings," *SIAM J. Comput.*, Vol. 6, 1977, pp. 323-350.

[KR-87] R.M. Karp and M.O. Rabin, "Efficient randomized pattern-matching algorithms," *IBM Journal of Research and Development*, Vol. 31, No. 2, March 1987, pp. 249-260.

[LD-91] A. Lupas, M. Van Dyke, J. Stock, "Predicting coiled coil from protein sequences," *Science* Vol. 252, 1990, pp. 1162-1164.

[LLE-91] R. Lüthy, A. D. McLachlan, and D. Eisenberg, "Secondary structure-based profiles: use of structure-conserving scoring tables in searching protein sequence databases for structural similarities," *Proteins,* Vol. 10, 1991, pp. 229-239.

[LV-89] G.M. Landau and U. Vishkin, "Fast parallel and serial approximate string matching," *Journal of Algorithms,* Vol. 10, No. 2, June 1989, pp. 157-169.

[MM-88] E. W. Myers and W. Miller, "Sequence Comparison with Concave Weighting Functions," *Bulletin of Mathematical Biology*, Vol. 50, 1988, pp. 97-120.

[MM-89] E. W. Myers and W. Miller, "Approximate matching of regular expressions," *Bulletin of Mathematical Biology*, Vol. 51, No 1, 1989, pp. 5-37.

[NW-69] S.B. Needleman and C.D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequences of two proteins," *J. Molecular Biology,* Vol. 48, 1969, pp. 443-453.

[S-74] P.H. Sellers, "On the theory and computation of evolutionary distance," *SIAM J. Appl. Math,* Vol. 26, No. 4, 1974, pp. 787-793.

[SK-83] D. Sankoff and J.B. Kruskal (editors), *Time Warps, String Edits, and Macromolecules: the Theory and Practice of Sequence Comparison,* Addison-Wesley, Reading, MA, 1983.

[SW-81] T. F. Smith and M. S. Waterman (1981) "Identification of common Molecular subsequences," *J. Mol. Biol.*, 147:195-197.

[Wa-89] M. S. Waterman (editor), *Mathematical Methods for DNA Sequences,* CRC Press, 1989.

[We-73] P. Weiner (1973) "Linear pattern matching algorithm," *Proc. 14 IEEE Symposium on Switching and Automata Theory*, 1-11.

[WF-74] R. A. Wagner and M. J. Fischer, "The string-to-string correction problem," *JACM,* Vol. 21, No 1, 1974, pp. 168-173.