# FINDING FLEXIBLE PATTERNS IN A TEXT - AN APPLICATION TO 3D MOLECULAR MATCHING


Marie-France Sagot [1,3]

Alain Viari [1]

Joël Pothier [1]

Henri Soldano [1,2]

[1] Atelier de BioInformatique
CPASO - URA CNRS 448
Section de Physique et Chimie de l'Institut Curie
11, Rue P. et M. Curie 75005 - Paris

[2] LIPN - Université Paris Nord
URA CNRS 1507
Avenue J.B. Clément 93430 - Villetaneuse

[3] Institut Gaspard Monge
Université de Marne la Vallée
2, rue de la Butte Verte 93160 - Noisy le Grand

e-mail : sagot@radium.jussieu.fr

fax : (33-1) 40-51-06-36



*Abstract :*

*Finding certain regularities in a text is an important problem in many areas, for instance in the analysis of biological molecules such as nucleic acids or proteins. In the latter case, the text may be sequences of amino acids or a linear coding of 3D structures, and the regularities then correspond to lexical or structural motifs common to two, or more, proteins. We first recall an earlier algorithm allowing to find these regularities in a flexible way. Then we introduce a generalized version of this algorithm designed for the particular case of protein 3D structures, since these structures present a few peculiarities that make them computationally harder to process. Finally, we give some applications of our new algorithm on concrete examples.*



**keywords** : cliques, multiple alignment, protein structural matching.

## Introduction

The main motivation for the new algorithm presented in this paper is that of finding the patterns common to a set of protein structures. This algorithm is an extension of an earlier one which finds all the words common to a set of protein sequences in a flexible way.

Both were designed for the multiple comparison and alignment of proteins, and for the systematic search of all repetitions in a text. The new version is used for multiple, as opposed to pairwise, comparison of protein structures. That is one of the most important differences between it and algorithms that also look for patterns in structures, using dynamic programming (Usha et al., 1986; Zuker et al., 1989; Orengo et al., 1990; Subbiah et al., 1993; Matsuo et al., 1993), or using geometric hashing techniques (Nussinov et al., 1991). Although this latter algorithm appears to be more general than ours in the sense that it is not limited to the matching of contiguous residues, it should be stressed that we can compare more than two proteins simultaneously. So far as we know, none of the other algorithms is able to work with more than two structures at the same time. Moreover, there seems to be a difficulty finding in the literature a clear definition of the concept of multiple comparison. Another important motivation for this work is therefore to provide one. Working with the maximal cliques of a similarity relation is a natural way to do that.

We give in section 1 a few definitions. Then we present the earlier algorithm in section 2 and the new one in section 3. We shall do so in an abstract way, leaving to section 4 some illustrations of their use. The reason is that, although the new algorithm was conceived with the specific goal of finding motifs common to protein 3-D structures, the way it generalizes the earlier version may appear interesting in itself. Indeed, it shows how the whole process of searching for all common words, in sequences or structures, rests on an operation of set intersection, where the sets possess certain well-defined properties that are preserved all along. The nature of these properties can be extensively varied, they can be formulated so as to include different definitions of flexible matching. Further extensions of the original algorithm can then be applied to other problems in computational biology and will be suggested in the conclusion.

## 1. General definitions

Let s be a string over an alphabet $\Sigma$. We wish to find words that are repeated one or more time in s. For practical purposes (where we work with two or more sequences or structures), s actually represents the concatenation of various strings. Searching for repeated words in s is then searching for words common to a set of strings.

**Notation 1.1 :** We call $r \in \Sigma^*$ a word in s if $s = u_1 r u_2$ with $u_1, u_2 \in \Sigma^*$. We note $|r|$ the length (or size) of r.

**Definition 1.1 :** By a repeated word in s we mean any string $r \in \Sigma^+$ such that $s = u_1 r u_2 = u'_1 r u'_2$ with $u_1, u_2, u'_1, u'_2 \in \Sigma^*$ and $|u_1| \neq |u'_1|$.

We also use the word repetitions.

***Example 1.1 :***

*s = cababdb, r = ab*

*s = cdabababef, r = abab*

However, we want to look for these repetitions in a more flexible way, which means that we are not going to look only for exact repetitions but also for approximate ones. Let then R be a relation on the symbols of the alphabet $\Sigma$. This relation will be reflexive, symmetric, but in general not transitive. As an example, R may be obtained by setting a threshold on a numerical similarity matrix. Within this context, the maximal cliques of the relation R and its 'degeneracy' are defined.

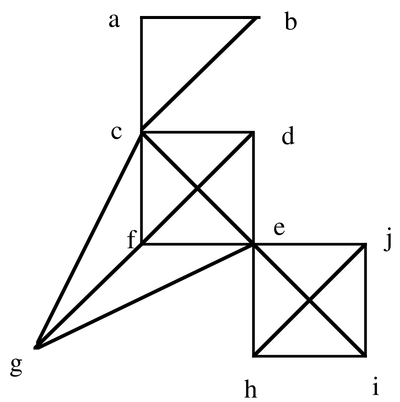
**Definition 1.2 :** A clique of a relation R on a finite set $\Sigma$ is a subset C of $\Sigma$ such that for all a, $b \in C$, we have a R b. C is said to be maximal if for any $c \in \Sigma$-C, $C \cup \{c\}$ is no longer a clique.

**Definition 1.3 :** Given a relation R on an alphabet $\Sigma$, we call g the maximal number of occurrences of a symbol of $\Sigma$ in all the maximal cliques of R.

In the special case where R is an equivalence relation or the identity relation, we have $g = 1$. We call this number g the '<u>degeneracy</u>' of R. In practice, for random strings, instead of g we can use its average value over $\Sigma$, noted $\bar{g}$.

***Example 1.2 :***

*Let $\Sigma = \{a, b, c, d, e, f, g, h, i, j\}$ and let R be the relation :*




*The maximal cliques of the relation R on $\Sigma$ are the sets :*

$c_1 = \{a, b, c\}$

$c_2 = \{c, d, e, f\}$

$c_3 = \{c, e, f, g\}$

$c_4 = \{e, h, i, j\}$

*and $g = 3$ (because c appears in three maximal cliques of R).*
*On the other hand, $\bar{g} = (1 + 1 + 3 + 1 + 3 + 2 + 1 + 1 + 1 + 1) \times \frac{1}{10} = 1.5$ (if all symbols have the same probability of appearing in strings).*

R provides a definition of 'approximate' matching between words :

**Definition 1.4 :** An <u>approximate repetition</u> of a word $u = u_1u_2...u_k \in \Sigma^+$ is a word $v = v_1v_2...v_k \in \Sigma^+$ such that $(u_i \; R \; v_i)$ for all $i \in \{1,2,...,k\}$ - we denote it by $(u \; R_k \; v)$.

***Example 1.3 :***

*Let $\Sigma$ and R be those of example 1.2.*

*Then in $s$ = faaihab, aa appears approximately repeated at positions 2 (aa) and 6 (ab).*

## 2. Earlier algorithm for finding all approximate repetitions in a string

The algorithm devised by Soldano, Viari and Champesme (Soldano et al., 1994) to solve the problem of finding all the approximate repetitions in a string is itself an extension of an older one presented by Karp, Miller and Rosenberg (Karp et al., 1972), which we call KMR.

KMR is a linear time algorithm that finds all exact repetitions in a string and, in a more general way, in an oriented and labeled structure such as a matrix or a tree. We shall concern ourselves here with the case of strings only. Let s be a string over an alphabet $\Sigma$. The two problems Karp and his colleagues proposed to solve were the following :

Problem 1 : identify the position of all the words of a certain fixed length k that appear repeated in s;

Problem 2 : find the length $k_{max}$ of the longest repeated word in s, and solve problem 1 for $k = k_{max}$.

The main idea of KMR rests on the definition of the following equivalence relation over the positions of the string.

**Definition 2.1 :** Given an alphabet $\Sigma$ and a string $s = s_1s_2...s_n \in \Sigma^n$, two positions $i, j \in \{1,...,n-k+1\}$ of s are said to be k-equivalent, noted $i\ E_k\ j$, if and only if the words of length k starting at these positions in s are identical.

Given this definition, problems 1 and 2 can be formulated again as the problem of finding the partition of $E_k$. The classes of $E_k$ that interest us are then those whose cardinality is greater than or equal to two. The algorithm is based on the construction of these partitions, starting with $E_1$, then using $E_1$ to build $E_2$, $E_2$ to build $E_4$, $E_4$ to build $E_8$, etc. Details of the algorithm can be found in the original paper (Karp et al., 1972), and will be briefly explained when we shall talk of the generalization proposed by Soldano et al. (Soldano et al., 1994).

The time complexity of KMR is O(nlogk) where k is the length of the repetitions looked for (problem 1) or the length of the longest repeated words (problem 2). The space complexity is O(n).

Let us now briefly recall the algorithm devised by Soldano et al. to extend the KMR algorithm to the case of approximate matching (Soldano et al. 1994). From now on, this algorithm will be referred to as KMRC (C stands for Clique).

Let us denote R a reflexive, symmetric but not necessarily transitive relation on Σ (that is, R is no longer an equivalence relation as in KMR). The following relation $R_k$ over the positions of a string generalizes the equivalence relation $E_k$ of KMR :

**Definition 2.2 :** Given an alphabet Σ and a string $s = s_1 s_2 \ldots s_n \in \Sigma^n$, two positions i, j ∈ {1,...,n-k+1} of s are said to be in <u>k-relation</u>, noted i $R_k$ j, if and only if i+p R j+p for all p ∈ {0,...,k-1} .That is, the words of length k of s starting at positions i and j of s are approximately repeated words  in the sense of definition 1.4 (see figure 1).

Let Σ and R be those of example 1.2.
Then in the string :

s : a b c  a b b  a b d
(positions: i under the first a, j under the second a, k under the third a)

we have :
i $R_3$ j
i $R_3$ k

but we do <u>not</u> have :
j $R_3$ k

**Figure 1 : Illustration of definition 2.2.**

In the same way as KMR's algorithm was based on the iterative computation of the classes of $E_k$, KMRC's algorithm is based on the iterative computation of the maximal cliques of $R_k$. The mechanism for such computations remains the same and rests on the following two lemmas :

**Lemma 2.1 :** Let $s = s_1...s_n \in \Sigma^n$. Let $a, b \in \{1,...,n\}$ with $b \leq a$, and $i, j \in \{1,...,n-(a+b)+1\}$.

Then :

$i \; R_{a+b} \; j$ if and only if $i \; R_a \; j$ and $i+b \; R_a \; j+b$.

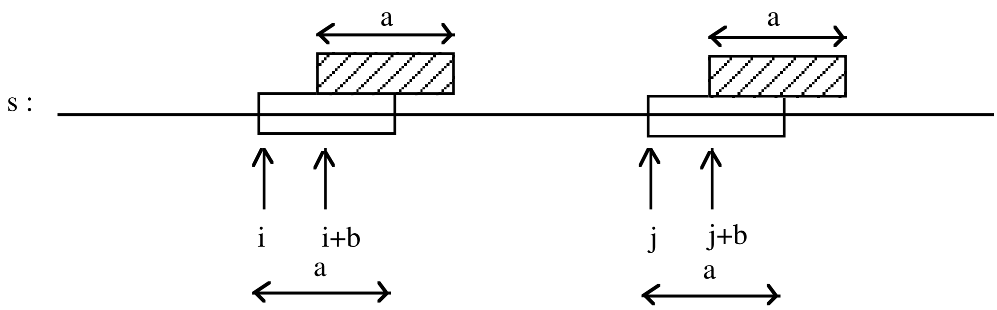



**Figure 2 : Illustration of lemma 2.1.**

**Lemma 2.2 :** Let $s = s_1...s_n \in \Sigma^n$. Let $a, b \in \{1,...,n\}$ with $b \leq a$, and $i, j \in \{1,...,n-(a+b)+1\}$.

Then :

$i \; R_{a+b} \; j$ if and only if $i \; R_a \; j$ and $i+a \; R_b \; j+a$.

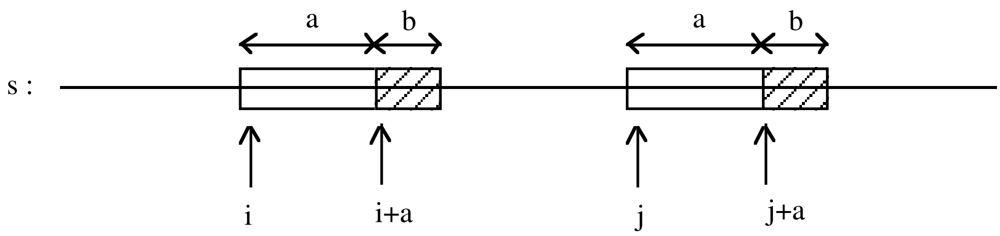



**Figure 3 : Illustration of lemma 2.2.**

In practice, lemma 2.1 or 2.2 is used with a = b to iteratively compute the words of size 2, 4, 8, etc. If k (or kmax) is not a power of 2, one should first find the words whose size k' < k is a power of 2 and then use lemma 2.1 to put the parts together in a final phase.

Before showing how the cliques of $R_k$ are calculated, let us introduce the notations and definitions that we shall use later.

**Notation 2.1 :** $C_X$ denotes a clique of the relation $R_k$ on a set X of objects of size k and c a clique of the relation R on $\Sigma$.

**Notation 2.2 :** Let $x \in \Sigma^k$ be a word of size k. x is denoted :

$$x = (x_1, x_2, ..., x_k)$$

with $x_i \in \Sigma$.

We shall now present certain propositions proven in Soldano et al., 1994. They will be given here without further comment, the proofs are to be found in the original article.

**Proposition 2.1 :** Let $s = s_1...s_n \in \Sigma^n$ be a string and $X = \{1,...,n\}$ be the set of positions in s. Let $\{\bar{c}_1,...,\bar{c}_e\}$ be the e maximal cliques of R and $\{C^1_X,...,C^e_X\}$ be the subsets defined by :

$$C^i_X = \{x \in X \ / \ s_x \in \bar{c}_i\}.$$

Then :

(i) $C^i_X$ is a clique of $R_1$

(ii) The set of maximal cliques of $R_1$ is included in $\{C^1_X,...,C^e_X\}$.

**Definition 2.3 :** Let I be a set of indices, $I_{+d}$ ($I_{-d}$) is the set obtained by adding (respectively, subtracting) the integer d to all indices in I.

**Proposition 2.2 (a) :** Let $C_X$ and $C'_X$ be two cliques of $R_a$. $(C'_X)_{-b}$ is the set obtained by subtracting b from all positions of $C'_X$ superior or equal to b, the others being discarded. Then :

$$C_X \cap (C'_X)_{-b} \text{ is a clique of } R_{a+b}.$$

**Proposition 2.2 (b) :** Let $C_X$ be a clique of $R_a$ and $C'_X$ a clique of $R_b$. $(C'_X)_{-a}$ is then the set obtained by subtracting a from all positions of $C'_X$ superior or equal to a, the others being discarded. Then :

$$C_X \cap (C'_X)_{-a} \text{ is a clique of } R_{a+b}.$$

**Proposition 2.3 (a) :** Let $C''_X$ be a maximal clique of $R_{a+b}$. Then there exist two maximal cliques $C_X$ and $C'_X$ of $R_a$ such that $C''_X = C_X \cap (C'_X)_{-b}$.

**Proposition 2.3 (b) :** Let $C''_X$ be a maximal clique of $R_{a+b}$. Then there exist a maximal clique $C_X$ of $R_a$ and a maximal clique $C'_X$ of $R_b$ such that $C''_X = C_X \cap (C'_X)_{-a}$.

These results show us how to calculate the maximal cliques of $R_k$, $k \geq 1$. Indeed, proposition 2.1 tells us that the set of maximal cliques of $R_1$ is obtained by first constructing the set $\mathcal{C} = \{C_X^1, \ldots, C_X^e\}$, then by eliminating the elements $C_X^i$ of $\mathcal{C}$ such that $C_X^i \subset C_X^j$ for all $C_X^i$ and $C_X^j$ belonging to $\mathcal{C}$.

Propositions 2.2 and 2.3 show that the construction of the relations $R_k$ and of their maximal cliques is basically an operation of set intersections of cliques of $R_{k'}$ ($k' < k$). More precisely, as in KMR, we use $R_1$ to compute $R_2$, $R_2$ to compute $R_4$, and so on. As in KMR, these intersections are implemented by using stacks of integers. More precisely, we need two arrays, P and Q, of stacks in order to sort the words of length k according to their labels (i.e. the numbers of the maximal cliques to which they belong), and so as to find the labels of the words of length 2k. These stacks contain the positions of the words in the string. We also need another array of stacks, $V_k$, which, for each position i in s, contains at step k the labels of the maximal cliques of $R_k$ to which the k-length word starting at position i belongs. The first steps of the algorithm are illustrated in figure 4. Finally, it should be mentioned that although the intersections performed give us cliques, these cliques are not always maximal. It is therefore necessary, at each step, to remove non-maximal cliques. This is done with additional tests of set inclusion which can be carried out efficiently (see Soldano et al., 1994).

$\Sigma = \{a,b,c,d\}$ ; maximal cliques of R = {{a,d},{a,c},{b}}

**1** **2** **3**

s =

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| a | a | c | b | d | a | c | b | d | d |

positions 1 2 3 4 5 6 7 8 9 10

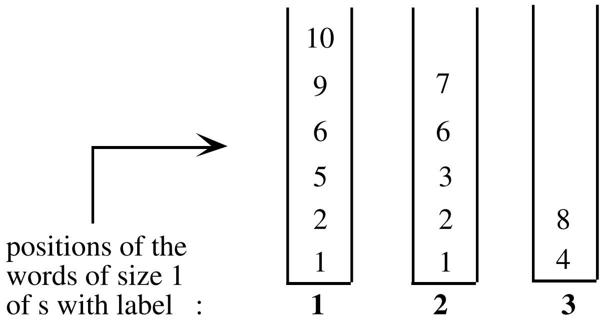
$V_1$ =

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **2** | **2** | | | | **2** | | | | |
| **1** | **1** | **2** | **3** | **1** | **1** | **2** | **3** | **1** | **1** |

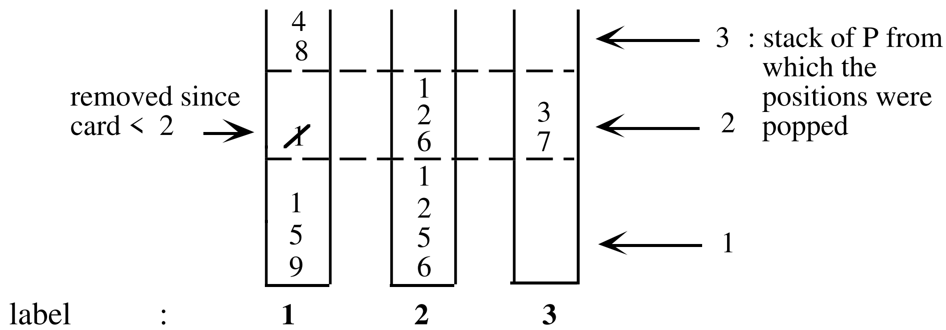
Labels of the words of size 1 of s given by the relation $R_1$

**Construction of $R_2$** : use of two arrays of stacks, P and Q

array P : sorts the positions of the words of size 1 of s by the labels they have in $R_1$ :
pop label j from each V[i] and push i into P[j].




array Q : sorts the elements of P by the labels of the word of size 1 located at the next position in s
pop position p from each P[i] and push it into all Q[$V_1$[p+1]] (note that p = 10 is discarded)




**Construction of $V_2$** : initialize a counter c to 1; each time p comes from a different stack of P (dashed line) or a different stack of Q, increment c; push c into $V_2$[p]

$V_2$ =

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **4** | | | | | | | | |
| **3** | **4** | | | **4** | **4** | | | |
| **2** | **3** | **5** | **1** | **2** | **3** | **5** | **1** | **2** |

So, {1,5,9}, {1,2,5,6}, {1,2,6}, {3,7} and {4,8} are the positions of all the approximate repetitions of size 2 of s.

Note however that since {1,2,6} is included in {1,2,5,6}, clique number 3 is not maximal and should be removed (see Soldano et al., 1994)

**Figure 4 : An illustration of how algorithm KMRC works, which uses lemma 2.1 or 2.2 with a = b = 1 to find all the approximate repetitions of size 2 of a string.**

Let k be the length of the approximate repetitions we wish to identify, or that of the longest approximately repeated words in the string being analyzed, the time complexity of KMRC is $O(n.g^{2k}.klogk)$. The total number of elements in all the maximal cliques of $R_k$ is bounded above by $n.g^k$ (see details in Soldano et al., 1994). The space complexity is $O(n.min(n.g^k, e^k))$, where e is the number of cliques of the relation R. It is important to note that these are worst-case scenarios. As we shall see later, $\bar{g}$ and k are small values in most biological applications (e.g. sequence alignments), typically between 1.1 and 1.2 for $\bar{g}$ and around 10-12 for k.

**3. Problem with KMRC and how to solve it - algorithm KMRCJ (KMRC with Jumps)**

The KMRC algorithm works just fine in most cases. But in certain special situations it either takes too long to perform its calculations, or it simply stops before reaching the end because of a memory 'explosion'.

The cases for which the algorithm presents a poor performance arise in the following circumstances :

1. the alphabet $\Sigma$ is 'big' (typically over a hundred symbols);
2. the relation R on $\Sigma$ presents an important degeneracy in comparison to an equivalence relation, that is $\bar{g}$ is 'big' (typically over 4);
3. the string s to be analyzed has the peculiarity of containing one symbol that repeats itself consecutively a great number of times, forming many words of a 'small' length.

It should be pointed out that problem 2 is not specific to the algorithm but comes from the fact that the number of solutions increases rapidly when $\bar{g}$ is big. As with problem 3 however, the 'explosion' is more considerable with 'small' words. Since the principle of the algorithm is to build words by increasing length (1, 2, 4, 8, ...), the overall algorithm may have to spend too much time and space building the 'small' words that will be used to build the 'longer' ones.

As an example, let us suppose that we wish to identify all the repeated words of length 8 of s. With KMR/KMRC we can only identify these repetitions of length 8 by first identifying all those of length 1, 2 and 4 which are <u>subwords</u> of the repeated words of length 8. If there are too many such subwords, the overall performance of the algorithm will be poor. It should however be stressed that nothing obliges us to consider these subwords of length 1, 2, 4 as 'real words', that is as a series of <u>contiguous</u> symbols. For instance, we could define two identical 'special words' of length 2 to be two 'real words' of length 8 whose first and last symbols are related by R.

Before we see why this idea is enough to solve our problem, we must prove its correctness by generalizing the lemmas and some of the definitions and propositions of KMRC. We start by defining a new relation on the positions of a string.

**Definition 3.1 :** Let $\Sigma$ be an alphabet, R a relation on $\Sigma$, I a subset of $\{0,...,n-1\}$ and $s = s_1...s_n \in \Sigma^n$. Two positions $i, j \in \{1,...,n-\max\{q \in I\}\}$ are said to be in <u>I-relation</u>, noted $i\ R_I\ j$, if and only if $i+m\ R\ j+m$ for all $m \in I$ (see figure 5).

Let R be the relation on {a,b,c,d,e,f} given by the maximal cliques : {a,d}, {a,c}, {b}, {e,f}

s : ________ a b c d ________ d b a a ________
i j

Here, we have $i\ R_{\{1,4\}} j$

**Figure 5 : Illustration of definition 3.1.**

Lemmas 2.1 and 2.2 of KMRC become then lemma 3.1 (see figure 6).

**Lemma 3.1 :** Let $s = s_1...s_n \in \Sigma^n$, $I, J \subset \{0,...,n-1\}$ and $i, j \in \{1,...,n-\max\{q \in I \cup J\}\}$. Then :

$i\ R_{I \cup J}\ j$ if and only if $i\ R_I\ j$ and $i\ R_J\ j$.

Let R be the relation on {a,b,c,d,e,f} given by the maximal cliques : {a,d}, {a,c}, {b}, {e,f}

s : ______ a b c d e f ______ a b c a f e ______

Lemma 3.1 states : $i\ R_{\{1,2,3,4,5,6\}}\ j \iff i\ R_{\{1,3,5\}}\ j$ and $i\ R_{\{2,4,6\}}\ j$

**Figure 6 : Illustration of lemma 3.1.**

Observe that lemmas 2.1 and 2.2 are special cases of lemma 3.1 with respectively ($I = \{0,...,a-1\}$, $J = \{b,...,a+b-1\}$, ($b \leq a$)) and ($I = \{0,...,a-1\}$, $J = \{a,...,a+b-1\}$).

**Notation (2.1)':** Let $I \subset \{0,...,n-1\}$. We call $C^{I}_{X}$ a clique of the relation $R_I$ on a set X of objects of size | I | and c a clique of the relation R on $\Sigma$.

**Notation (2.2)' :** Let I be a subset of $\{0,...,n-1\}$ with | I | = k. Let $x \in \Sigma^k$ be a word of size k. x is denoted :

$$x = (x_{i_1}, x_{i_2}, ..., x_{i_k}) \text{ with } i_j \in I \text{ and } x_{i_j} \in \Sigma \text{ for all } j \in \{1,...,k\}.$$

**Proposition (2.2)' :** Let I, J be subsets of $\{0,...,n-1\}$. Let $C_X$ be a clique of $R_I$ and $C'_X$ a clique of $R_J$. Then $C_X \cap C'_X$ is a clique of $R_{I \cup J}$.

**Proof :**

For all $x, y \in C_X \cap C'_X$, we have $x\ R_I\ y$ since $x, y \in C_X$ and $x\ R_J\ y$ since $x, y \in C'_X$. Then, by lemma 3.1, $x\ R_{I \cup J}\ y$ and $C_X \cap C'_X$ is a clique of $R_{I \cup J}$.

**Proposition (2.3)' :** Let I, J be subsets of $\{0,...,n-1\}$. Let $C''_X$ be a maximal clique of $R_{I \cup J}$. Then there exist a maximal clique $C_X$ of $R_I$ and a maximal clique $C'_X$ of $R_J$ such that $C''_X = C_X \cap C'_X$.

**Proof :**

For all $x, y \in C''_X$, we have $x\ R_{I \cup J}\ y$ and thus, by lemma 3.1, $x\ R_I\ y$ and $x\ R_J\ y$. $C''_X$ is therefore a clique of $R_I$ and a clique of $R_J$. There exist then a maximal clique $C_X$ of $R_I$ and a maximal clique $C'_X$ of $R_J$ such that $C''_X \subset C_X \cap C'_X$. But, by hypothesis, $C''_X$ is maximal. Then, $C''_X = C_X \cap C'_X$.

Up to now, we have done nothing but modify the results of KMRC in a way that broadens them. Here is a new proposition that results from lemma 3.1 and from propositions (2.2)' and (2.3)' :

**Proposition 3.1 :** Let $s \in \Sigma^n$ and let i, j be positions in s. Then :

$$i \; R_k \; j \quad \text{if and only if} \quad i \; R_{I_1} \; j \quad \text{and} \quad i \; R_{I_2} \; j \quad \text{and ... and} \quad i \; R_{I_m} \; j$$

with :

$I_1, I_2, \ldots, I_m$ subsets of $\{0,\ldots,k\text{-}1\}$ such that $\bigcup_{p=1}^{m} I_p = \{0,\ldots,k\text{-}1\}$ and $I_p \not\subset I_q$ for all p, $q \in \{1,\ldots m\}$, $p \neq q$.

**Corollary 3.1 :** Let $s \in \Sigma^n$ and let i, j be positions in s. Then :

$i \; R_k \; j$ if and only if $i \; R_{\{0\}} \; j$ and $i \; R_{\{1\}} \; j$ and ... and $i \; R_{\{k\text{-}1\}} \; j$.

Furthermore, the order in which the intersections of the relations $R_{\{p\}}$ of corollary 3.1 are performed is clearly indifferent. We give below an example of this corollary.

***Example 3.1 :***

*Let Σ = {a,...z} and let us suppose, for simplicity's sake, that R = identity.*

*Let* *s* = ---------------*abcdefgh*---------------*abcdefgh*-------------

*with* *pos* = *i* *j*

*Then* $i\ R_8\ j$ *because :*

- $i\ R_{\{0\}}\ j$ :

*s* = -------------**a**bcdefgh---------------**a**bcdefgh-------------

*and* - $i\ R_{\{7\}}\ j$ :

*s* = -------------**a**bcdefg**h**---------------**a**bcdefg**h**-------------

*and* - $i\ R_{\{1\}}\ j$ :

*s* = -------------**ab**cdefg**h**---------------**ab**cdefg**h**-------------

*and* - $i\ R_{\{6\}}\ j$ :

*s* = -------------**ab**cdef**gh**---------------**ab**cdef**gh**-------------

*and* - $i\ R_{\{2\}}\ j$ :

*s* = -------------**abc**def**gh**---------------**abc**def**gh**-------------

*and* - $i\ R_{\{5\}}\ j$ :

*s* = -------------**abc**de**fgh**---------------**abc**de**fgh**-------------

*and* - $i\ R_{\{3\}}\ j$ :

*s* = -------------**abcd**e**fgh**---------------**abcd**e**fgh**-------------

*and* - $i\ R_{\{4\}}\ j$ :

*s* = -------------**abcdefgh**---------------**abcdefgh**-------------

The following algorithm will then find all the approximately repeated words of length k of a string s using proposition 3.1. It is illustrated in figure 7.

/* *Identification of all the approximately repeated words of length k.*

/* *Input : Sets $I_1,...,I_m$ with :*

/* $I_1, I_2,..., I_m$ *subsets of* $\{0,...,k-1\}$ *such that* $\bigcup_{p=1}^{m} I_p = \{0,...,k-1\}$ *and* $I_p \not\subset I_q$ *for all*

/* $p, q \in \{1,...m\}, p \neq q$.

/* *Observation: We can determine the cliques of* $R_{I_p}$ *for all* $p \in \{1,...,m\}$, *either from the*

/* *cliques of R, or from the cliques of* $R_{I_J} = \bigcap_{j \in J} R_{I_j}$ *(where* $J \subset \{0,...,k-1\}-\{p\}$ *and* $R_{I_J}$

/* *will have been constructed in a previous step.*

construct the relation $R_1$ and its maximal cliques as in KMRC;

iteratively construct the relations $R_J = R_{I_p} \cap R_{I_q}$ where $I_p \not\subset I_q$ and $I_q \not\subset I_p$ and $J = I_p \cup I_q$ $\subset \{0,...,k-1\}$, and determine the cliques $C_X^J$ knowing that $C_X^J = C_X^{I_p} \cap C_X^{I_q}$ where $C_X^{I_p}$ and $C_X^{I_q}$ are the maximal cliques of $R_{I_p}$ and $R_{I_q}$ respectively (calculated in a previous step), and $p, q \in \{1,...,m\}$;

after that apply the tests of set inclusion as in KMRC

if $J = \{0,...,k-1\}$

stop

**Algorithm 3.1 : Algorithm KMRC modified.**

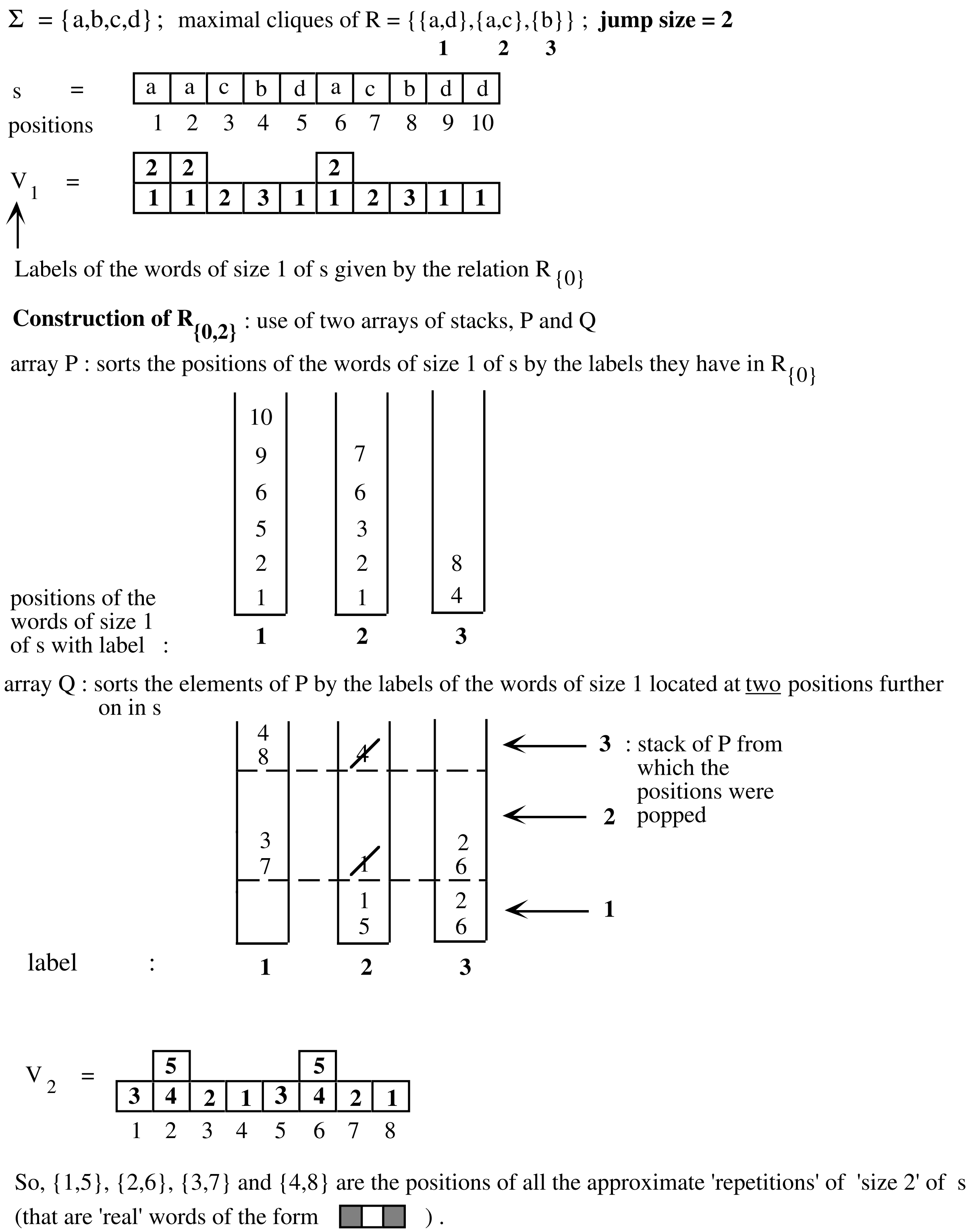



**Figure 7 : An illustration of how algorithm KMRCJ works, to find all the repeated 'words' of 'size' 2 of a string, using lemma 3.1 with the sets {0} and {2}.**

If we wish to use corollary 3.1 instead of proposition 3.1, the algorithm will then simplify to the one given below :

/* *Identification of all the approximately repeated words of length k.*

/* *Input : A set {0,...,k-1} of indexes placed in a certain order :* $<j_0,...,j_{k-1}>$.

construct the relation $R_1$ and its maximal cliques $C_X^i$ as in KMRC, store the information

concerning them;

for every $j_i$ of $<j_0,...,j_{k-1}>$ picked in order :

construct the relations of $R_J = R_I \cap R_{\{j_i\}}$ where $R_I = \bigcap_{m \in \{j_0,...,j_{i-1}\}} R_{\{m\}}$

and $J = I \cup \{j_i\}$, and determine the cliques $C_X^J$ knowing that $C_X^J = C_X^I \cap C_X^{\{j_i\}}$,

where $C_X^I$ are the maximal cliques of $R_I$ and $C_X^{\{j_i\}}$ those of $R_1$ (they are the $C_X^i$

calculated in the beginning);

after that apply the tests of set inclusion as in KMRC

**Algorithm 3.2 : Algorithm KMRC modified - Special case of algorithm 3.1.**

These algorithms allow us to solve our problem by jumping along the string s. This is the reason why we call them KMRCJ (J from Jump). Note however that the jumps are not performed in a haphazard way (this is illustrated in example 3.1 above).

Let us suppose again we want to find all the repeated words of length 8. If we follow all the steps shown in example 3.1 - that is, if we calculate $R_{\{0\}}$, then $R_{\{7\}}$, $R_{\{1\}}$, $R_{\{6\}}$, etc - it is clear that, once the algorithm reaches the point where it is trying to identify the repeated words of length 8, the 'special' words have become 'real' words again. Furthermore, the algorithm is able to identify all of them, no information is lost for the purpose we had in mind (finding the 'real' repeated words of length 8). Finally, and that is what really matters, the new algorithm may find far less repeated words of a smaller length. This is particularly true when the string is biased and exhibits a lot of 'small' repeated words. Example 3.2 below illustrates this point.

***Example 3.2 :***

*Let us suppose Σ = {a, b, c, d, e, f, g, h, i, j, k, l} and R = identity. Let s be the string given below :*

*s = ... ~~a a a a~~ | i j k l ... a a a a | e f g h ... a a a a | e f g h ...*

*If we were trying to identify the repeated words of length 8 of s with KMRC, the first word aaaaijkl would be eliminated as a possible candidate to be a repeated word of aaaaefgh at step k = 8 only.*

*With KMRCJ, this word is eliminated at step k = 2 since a = a but l ≠ h.*

It is possible to jump in many ways other than the one illustrated in example 3.1. However in our implementation, that is the one we adopted.

The time complexity of KMRCJ compared to that of KMRC depends on the kind of jump made, but in all circumstances concerns only the factor logk, where k is either a fixed length, or that of the longest approximate repetitions. In the worst possible case (if k is close to the size of the jump we wish to perform), that complexity is $O(n.g^{2k}.k^2)$.

The space complexity remains the same as in KMRC. Note that once again these are worst-case complexities.

## 4. Biological applications of KMRC and KMRCJ

An immediate application of KMR concerns the case where the string s corresponds to the biological 'sequence' of a protein or to the concatenation of a set of such biological sequences. Landraud, Avril and Chretienne (Landraud et al., 1989) have shown that KMR may constitute the basis of a multiple alignment algorithm (called 'alignment by blocks'). Given a set of protein sequences, the procedure is the following : first search for the longest word that is repeated in the sequences of the set, 'cut' this word out from the sequences and operate the recursion on the parts that remain to the left and right. A shortcoming of this approach comes from the use of KMR itself, which limits the blocks to being exact repeats. Therefore words like AGFI and AGYL appear to be different although from a biological point of view they should be considered as similar. Replacing KMR by KMRC in Landraud's algorithm, thus broadening its applicability, is quite simple. Relation R between the amino acids may be easily obtained by setting a threshold on a 'classical' similarity matrix such as PAM250 (Dayhoff et al., 1972). Our experience in this field has proven that the value $\bar{g}$ is usually very low (close to 1.2) and that, except in pathological cases (highly repeated sequences), KMRC behaves much like KMR. There is therefore no real need here for KMRCJ (Viari, 1993).

A second application, on which we shall focus now and which motivated the elaboration of KMRCJ, will exemplify the usefulness of this algorithm on a particular aspect of the analysis of biological molecules. More specifically, the problem that concerns us is to find common substructures in a set of protein 3D structures. This problem has already been addressed by several authors (Usha et al., 1986; Zuker et al., 1989; Orengo et al., 1990; Nussinov et al., 1991; Subbiah et al., 1993; Matsuo et al., 1993). One of the possible approaches is to use the internal coordinates of the protein backbone as a new alphabet and to search for an optimal 'local' alignment between the two structures with a dynamic programming algorithm. However, this method can be applied to <u>two</u> proteins only and, as far as we are aware, no solution has yet been proposed for a larger set of molecules. Before explaining our own approach, we shall give some biological background.

The 3D structure of a protein is determined by the spatial arrangement of the atoms of its amino acids. These amino acids are linked together in a chain called the polypeptidic chain. This chain folds itself in space, sometimes producing regular forms such as helices and sheets (Creighton, 1993). The representation we give for such a structure preserves the linear order of the residues along the polypeptidic chain, and that is why we can consider working with strings. When studying such a structure, it is often possible to ignore the lateral chain of each residue and to focus attention on the backbone only. It is well known (Creighton, 1993) that the local conformation of this backbone can be defined, at each residue, by three internal coordinates usually referred to as three dihedral angles : $\Phi$, $\Psi$ and $\omega$ (Ramachandran et al., 1963). Let us remark that other internal coordinates systems may be used as well (Levitt, 1976). For chemical reasons, $\omega$ is fixed (to 0 or 180°) and can usually be forgotten, and the internal coordinates of the backbone are usually represented on a two-dimensional map $\Phi$, $\Psi$ called a Ramachandran map (Ramachandran et al., 1963) (see figure 8). Finally, the structure of the backbone can be uniquely defined by the linear succession of the pairs of angles ($\Phi$, $\Psi$) along the backbone (it should be noted that a rigorous definition of $\Phi$ and $\Psi$ actually involves two residues but each pair can be assigned to one position only). The problem is that these pairs of angles represent pairs of real values, and we would like to work with discrete symbols. We must then find a way to recode these pairs into discrete values. In order to do it, we construct a grid of mesh $\varepsilon$° on the Ramachandran map (see figure 8). What is shown in the figure as flat is actually the surface of a sphere, hence all the following considerations about squares on this map will implicitly assume that these squares actually wrap around the edges. The center of each small square becomes a node of a square lattice. What is important for our purpose is that each node of the lattice corresponds to a symbol of a new alphabet $\Sigma$. Any real valued pair of angles is thus coded into the symbol represented by the small square inside which the pair is plotted. A relation R is then defined between the new symbols (nodes) of the map in the following way :

$$\forall (\alpha, \beta) \in \Sigma^2,\ \alpha \mathrel{R} \beta \quad \text{if and only if } \exists \text{ a square of side } 2K\varepsilon \text{ enclosing } \alpha \text{ and } \beta.$$

In the preceding definition, $\alpha$ and $\beta$ are nodes of the lattice and K is a parameter (called margin) that can be adjusted to broaden or narrow the matching precision. In terms of angles, the previous definition simply means that two pairs of angles match if they are equal plus or minus ($\Delta\Phi = K\varepsilon$, $\Delta\Psi = K\varepsilon$). But it is important to note that R is actually defined on the set of nodes of the lattice, not on the pairs of angles themselves. Finally, let us remark that <u>the maximal cliques of R on $\Sigma$ are, by definition, all (big) squares of side $K\varepsilon$ centered on each node</u>. Therefore, one can index each maximal clique of R by its center on the map. The alphabet may contain many symbols (e.g. 5182 symbols for a mesh of 5°) and the degeneracy of the relation is usually rather high (e.g. g = 9 for K = 1).

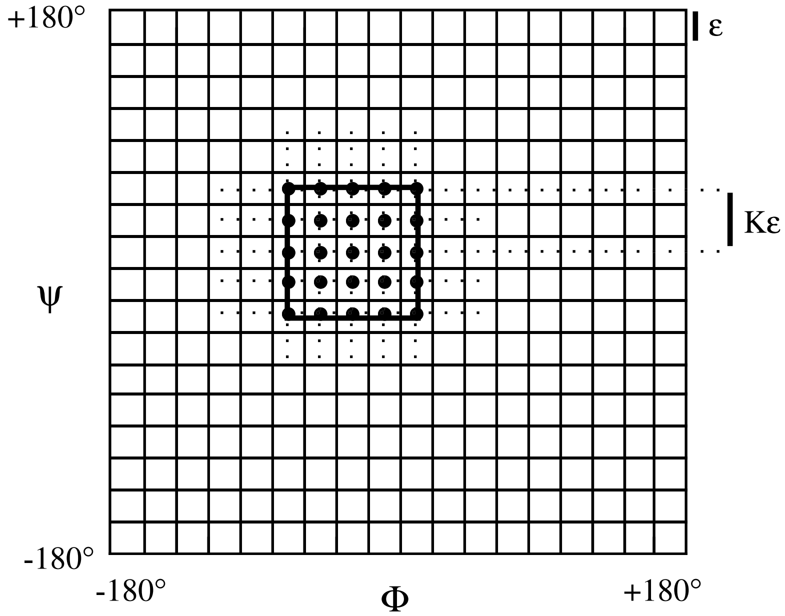



**Figure 8 : Sampled Ramachandran map :
one square of side ε = one symbol of the alphabet**

In order to find the 'structural words' that are approximately repeated in a string over this alphabet we have just established, we can use the KMRC algorithm. But as mentioned above, the structure of a protein may present certain regular forms such as helices and sheets. The pairs of angles inside these helices are all identical ($\Phi \approx -40°$ and $\Psi \approx -70°$ for $\alpha$-helices). They are therefore all coded into the same symbol. The structure of a protein containing a great number of such helices is thus coded by a string showing exactly the characteristics mentioned at the beginning of section 3, namely it exhibits contiguous repetitions of the same symbol. In most cases however, KMRC is able to find all the repeated 'structural words' of a protein or a small set of them in a quite efficient way. But in some cases (typically for proteins too rich in helices, or too long), it performs poorly, or simply cannot go to the end of its calculation because it finds too many repeated words of 'small' sizes (typically 4). KMRCJ was primarily designed to solve this problem. Before showing how KMRCJ achieves this goal, we wish to illustrate the algorithm on two examples. These are shown on figures 9 and 10. In the first case, three structures of proteases (bovine $\beta$-trypsin (1TLD), streptomyces proteinase A (2SGA) and porcine elastase (3EST)) were extracted from the Brookhaven crystallographic data bank (Bernstein et al., 1977; Abola et al., 1987). KMRCJ was run on this set with the following parameters: mesh = 5°, margin = 2. Two sets of 'structural words' of length 8, are then found on all three proteins. One of these sets is presented on figure 9-a as the succession of the corresponding $\alpha$-carbons (the $\alpha$-carbon of a residue is the backbone atom to which the lateral chain is connected, notice that an 8-length word of ($\Phi$, $\Psi$) actually corresponds to 10 matching $\alpha$-carbons). Then the three segments of figure 9-a are further superimposed (on the first segment) to emphasize their structural similarity (see figure 9-b). Finally the actual matching ($\Phi$, $\Psi$) pairs trajectories for the three segments are plotted on the Ramachandran map in figure 9-c.

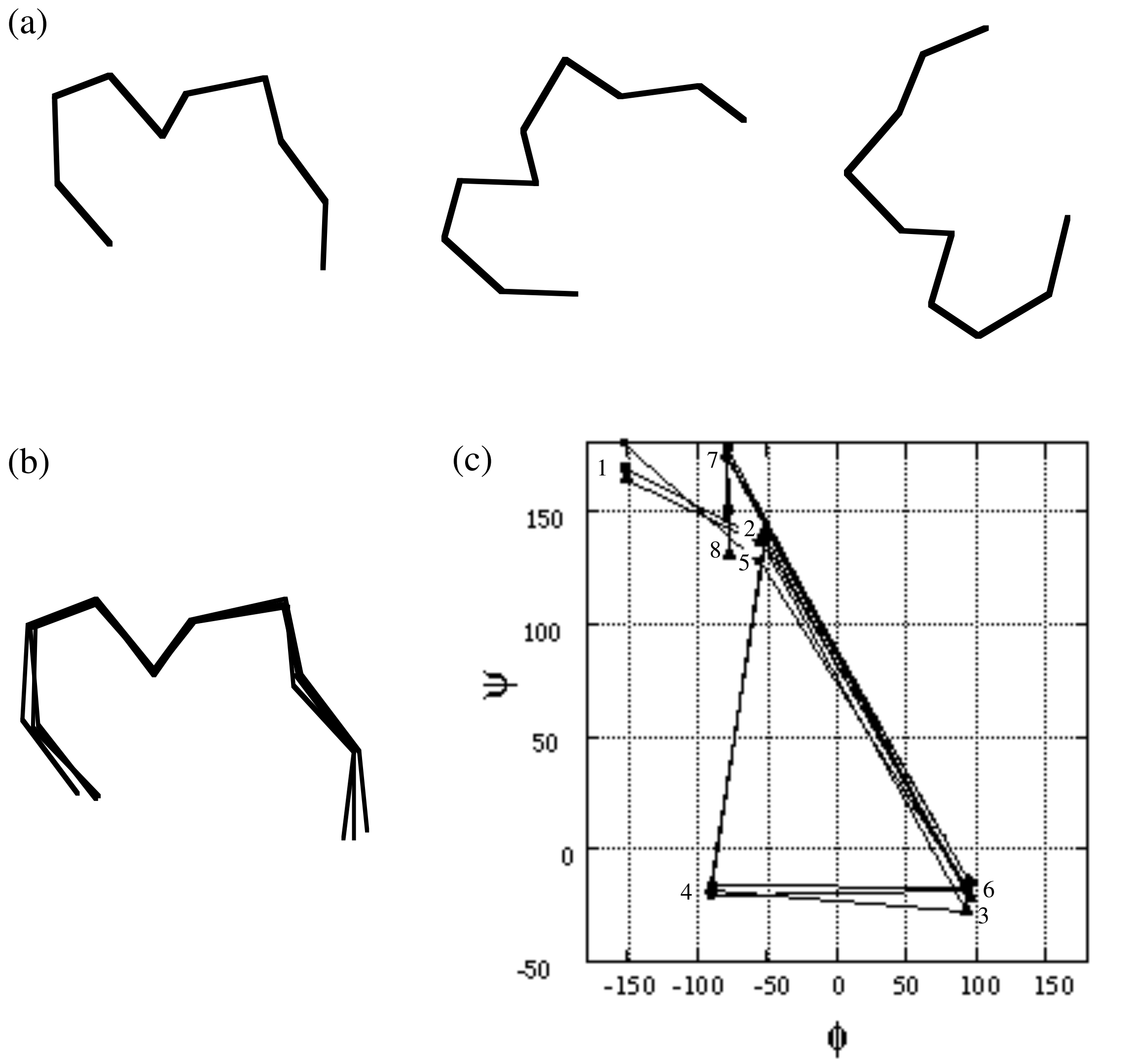



**Figure 9 : An example of 'structural words' found by KMRCJ in three proteases.**
**(a) $\alpha$-carbons backbones of the three segments.**
**(b) same as in -a-, superimposed on the first one.**
**(c) ($\Phi$, $\Psi$) trajectories on the Ramachandran map.**

Figure 10 shows a more complete example of the use of KMRCJ. Three structures of Cytochromes P450 (P450-TERP (1CPT); P450-BM3 (2HPD); P450-CAM (3CPP)) were extracted from the Brookhaven data bank. The figure displays some of the 'structural words' common to the proteins found by the program (for clarity only a few of the words are shown). When the three global structures (dashed lines) are aligned on the central L-shaped helix, one can notice that other words are located in the same areas of the proteins.

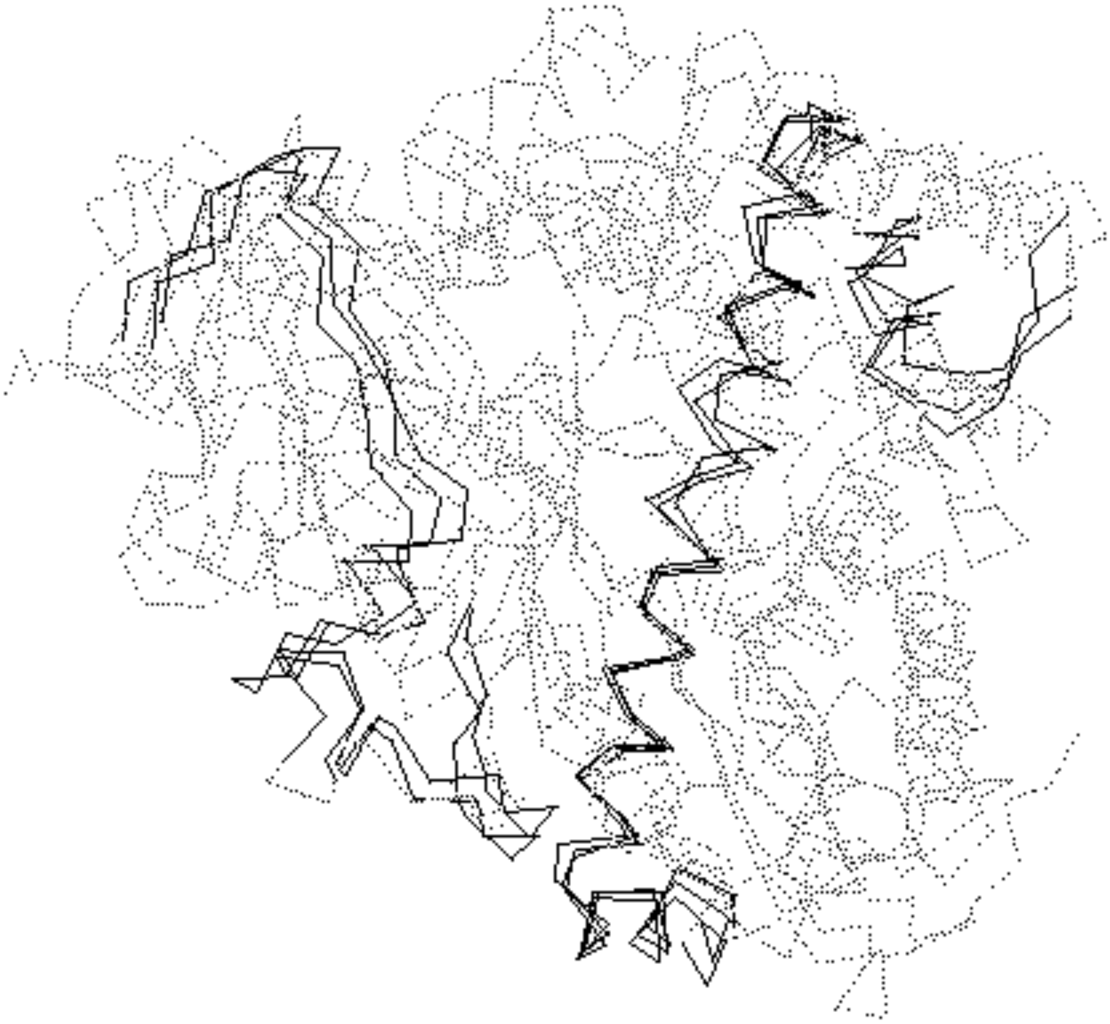


**Figure 10: Some of the 'structurals words' (bold) found on three Cytochromes P450. The whole proteins backbones were superimposed on the central L-shaped helix.**

This picture is intended to illustrate the notion of multiple alignment. Indeed, this is exemplified by the three words on the left side which are common to the three structures. In two of the structures (1CPT and 3CPP), these words are extended by an additional one at the bottom, whereas this extension is absent in the third structure (actually, the local structure of 2HPD in that area is sligthtly different and presents an additional loop). Other experiments on several protein structures are currently under way in our laboratory.

We shall illustrate now the advantage of KMRCJ versus KMRC in the context of protein structural matching. 46 non redundant protein 3D structures were extracted from the Brookhaven crystallographic data bank; then KMRC and KMRCJ were run on each one of the proteins duplicated once (this was just for the sake of testing) using the parameters : mesh = 3°, margin = 2. The execution times for KMRC vary from a few seconds to several minutes (on a Sparc 2 workstation). A more precise analysis shows that the execution times are not simply related to the length of a protein but also to its richness in $\alpha$-helices. In most cases KMRCJ runs much quicker, indeed the ratio between the KMRC time and the KMRCJ time ranges from 0.63 to 15. It should be pointed out that the gain is highest where KMRC is slowest. This is illustrated in figure 11 where this ratio is plotted against KMRC time.

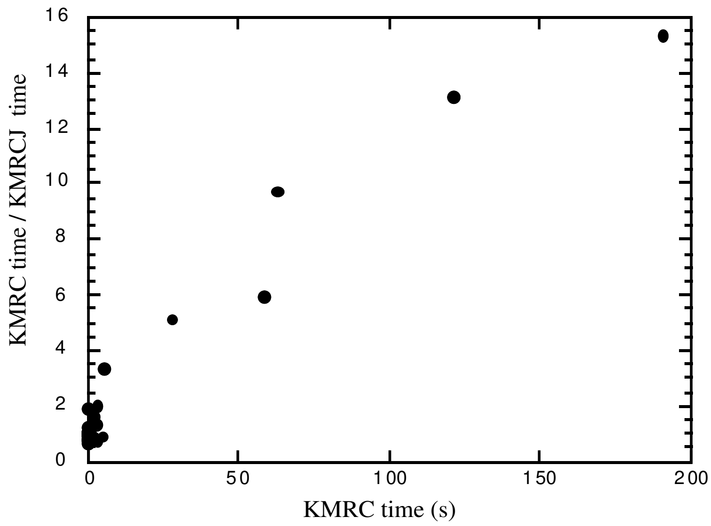



**Figure 11 : The ratio KMRC time/KMRCJ time plotted against KMRC time .**

## 5. Conclusion and future work

We have just shown an example where our new algorithm is able to find all the repetitions in a text - in that case, a linear coding of the 3-D structures of a set of proteins - in a way that is both flexible and efficient when the concerned text raises certain special computational difficulties. We stress the fact that for this application the use of the concept of maximal cliques is crucial since we want to introduce approximate matching between angles, and because it gives a clear definition of multiple matching between words. What we have shown also is that finding these repetitions can be seen as equivalent to realizing operations of intersection on sets possessing certain properties. The generalizations we had to make of our earlier results and algorithms can certainly be further developed to suit other needs. For instance, in the situations presented here, the property preserved, and hence the relation we worked with, was unique. An interesting extension to that is to deal with the case where there is not only one relation defined over the alphabet but a set of them. Soldano and his colleagues suggested (Soldano et al., 1994) that we can use a collection of relations defined over a universe of structured objects (such as strings). We can, for example, have a number n+1 of such relations, $R_0$, $R_1$, $R_2$,..., $R_n$, so that, for all objects x and y, we have :

$$x\ R_0\ y \Leftrightarrow x\ R_1\ y \text{ and } x\ R_2\ y \text{ and ... and } x\ R_n\ y.$$

That can give us a complete family of KMR(X) algorithms, each adapted to a particular problem, and all of them able to make comparisons on a multiple basis. Work is actually under way on such generalizations, that can treat mismatches and gaps on protein and acid nucleic acid sequences, and can perform the analysis of protein sequences and structures simultaneously.

Another problem still to be solved is that of establishing the average-case complexities of KMRC and KMRCJ.

Finally, one may observe that the almost linear behavior of KMRC (when working on sequences and on a relation derived from a biological similarity matrix) allows us to consider the possibility of scanning sequence databases from a multiple alignment point of view.

**Acknowledgments**

This work was supported by a grant from Organibio - Programme CM2AO.
The authors would like to thank Pascale Jean (URA CNRS 400 - St. Pères) for her help with the cytochromes P450.

## References

Abola, E. E.; Bernstein, F. C.; Bryant, S. H.; Koetzle, T. F.; Weng, J. (1987) Protein data bank, *Crystallographic Databases - Information Content, Software Systems, Scientific Applications,* eds. Allen, F.H.; Bergerhoff, G.; Sievers, R., Data Commission of the International Union of Crystallography, Bonn/Cambridge/Chester, pages 107-132.

Altschul, S.F.; Gish, W.; Miller, W.; Myers, E.W.; Lipman, D.J. (1990) Basic local alignment search tool, *J. Mol. Biol.*, **215**, 403-410.

Bernstein, F. C.; Koetzle, T. F.; Williams, G. J. B.; Meyer, Jr, E. F.; Brice, M.D.; Rodgers, J. R.; Kennard, O.; Shimanouchi, T.; Tasumi, M .(1977) The protein data bank: a computer-based archival file for macromolecular structures, *J. Mol. Biol.*, **112**, 535-542.

Creighton, T.E. (1993) *Proteins, Structures and Molecular Properties*, Freeman.

Crochemore, M.; Rytter, W. (1991) Usefulness of the Karp-Miller-Rosenberg algorithm in parallel computations on strings and arrays, *Theoret. Comput. Sci.*, **88**, 59-82.

Dayhoff, M.O.; Eck, R.V.; Park, C.M. (1972) *Atlas of Protein Sequence and Structures*, National Biomedical Research Foundation (Washington, D.C.), volume 5, supplement 3, pages 89-99.

Karp, R.M.; Miller, R.E.; Rosenberg, A.L. (1972) Rapid identification of repeated patterns in strings, trees and arrays, *Proc. 4th Annu. ACM Symp. Theory of Computing*, pages 125-136.

Landraud, A.; Avril, J.F.; Chretienne, P. (1989) An algorithm for finding a common structure shared by a family of strings, *IEEE Trans. on Pattern Analysis and Machine Intelligence*, **11**, 890-895.

Levitt M. (1976) A simplified representation of protein conformations for rapid simulation of protein folding. *J. Mol. Biol.,* **104**, 59-107.

Matsuo, Y.; Kanehisa, M. (1993) An approach to systematic detection of protein structural motifs, *CABIOS.* **9**, 153-159.

Nussinov, R.; Wolfson, H.J. (1991) Efficient detection of three-dimensional motifs in biological macromolecules by computer vision techniques, *Proc. Natl. Aca. Sci. USA*, **88**, 10495-10499.

Orengo, C.A.; Taylor, W.R. (1990) A rapid method of protein structure alignment, *J. theor. Biol,* **147**, 517-551.

Pearson, W.R. (1990) Rapid and sensitive sequence comparison with FASTP and FASTA, *Methods in Enzymology 183*, Academic Press, pages 63-98.

Ramachandran, G.N.; Ramakrishnan, C.; Sasisekharan, V. (1963) Stereochemistry of polypeptide chain configurations, *J. Mol. Biol.*, **7**, 95-99.

Soldano, H.; Viari, A.; Champesme, M. (1994) Searching for flexible repeated patterns in labeled objects using a non transitive similarity relation, *Pattern Recognition Letters (in press).*

Subbiah, S.; Laurents, D.V.; Levitt, M. (1993) Structural similarity of DNA-binding domains of bacteriophage repressors and the globin core, *Current Biology.*, **3**, 141-148.

Usha, R.; Murthy, M.R.N. (1986) Protein structural homology : a metric approach, *Int. J. Peptide Protein Res.*, **28**, 364-369.

Viari A. (1993) SmartMulti: a tool for the multiple alignment of protein sequences using flexible blocks, *manuscript.*

Zuker, M.; Somorjai, R.L. (1989) The alignment of protein structures in three dimensions, *Bull. Math. Biol.,* **51**, 55-78.