1

Pattern Inference under many Guises

Marie-France Sagot
Yoshiko Wakabayashi

Abstract

This paper surveys some of the main combinatorial methods for inferring patterns from a string, or a set of strings. The types of problems that will be addressed are *repeat identification* and *common pattern inference*. The strings that will concern us represent biological entities, nucleic acid and protein sequences or, in some cases, structures. As is well-known, exact ("identical") patterns hardly make sense in biology; we consider here two types of similar ("non-identical") patterns. One comes from looking at what "hides" behind each letter of the DNA/RNA or protein alphabet while the other corresponds to the more familiar notion of "errors". The errors concern mutational events that may affect a molecule during DNA replication. Those that will be of interest to us are point mutations, that is, mutations operating each time on single letters of a biological sequence: substitution, insertion or deletion. Various notions of "similarity" are discussed, as well as other constraints the patterns may have to satisfy depending on the biological situation one is facing. Algorithms for the various notions, together with complexity analyses, are presented in some detail.

1.1 Introduction

In a world of constant changes, conserved patterns of any kind are objects of interest for various reasons. Some are prosaic: if one has to perform a given operation on a set of objects and some of these objects are identical, or almost identical, one may sometimes economize by performing just one operation for each group of identical objects, and perhaps by adjusting the operation to a smaller number of steps than starting from scratch for the group of almost identical objects.

Another reason for being interested in conserved patterns is deeper: things that do not change, or change a little, or less than others, are objects upon which strong constraints are potentially acting. The chances are great that these objects may perform a function, possibly an important one.

What aspects or properties of a pattern are preserved and how conserved they must be depends on the area of investigation one is concerned with. In many cases, this is a matter of debate even among specialists of the area.

The area that will interest us in this paper is molecular biology. Among the objects which may model biological entities, we shall consider strings only. Strings may correspond, among others, to nucleic acid sequences (DNA or RNA) or protein sequences, and, in some cases, structures. Another kind of object that is important for describing biological entities or modelling biological problems are graphs, sometimes directed, sometimes labeled at the nodes and/or edges. A special type of graphs, the trees, are used to represent, among others, phylogenies and some macromolecular structures (*e.g.* RNA secondary structures). We shall not treat graphs in this paper; they may be used but only as helps in some definitions.

We concentrate upon one aspect related to pattern conservation; namely, given one or more strings identify all patterns that have conserved some well-defined properties. This is called "inferring patterns". The properties will vary depending on the biological problem. Without going into detailed discussion on the biological subtleties, the properties are presented, specially in what they may influence the inference. Combinatorial methods for performing such inference are then surveyed (references to statistical methods for solving the same problems may be found in [34] and [45]). These include other persons work as well as our own.

1.2 Biological motivation

As is by now well known, biological sequences, whether DNA, RNA or proteins, may be represented as strings over an alphabet of 4 letters (DNA/RNA) or 20 (proteins). Some of the basic problems encountered in classical text analysis have their counterpart when the texts are biological, among them pattern matching. However, this problem, as well as others, comes with a twist once we are in the realm of biology: exact patterns hardly make sense in this case.

By *exact* above, we mean *identical*; and there are in fact at least two types of "non-identical" patterns one must consider in biology. One comes from looking at what "hides" behind each letter of the DNA/RNA or protein alphabet while the other corresponds to the more familiar notion of "errors". The errors concern mutational events that may affect a molecule during DNA replication. Those that will be of interest to us in this paper are *point mutations*, that is, mutations operating each time on single letters of a biological sequence: *substitution*, *insertion* or *deletion*. Considering substitutions only will sometimes be enough for dealing with a given problem.

Besides this, another important difference exists between classical text analysis and biological sequence analysis. This concerns the fact that the most interesting question is often not which specific, known, pattern has matches in one or more strings, but which patterns, unknown at start, match the string(s) enough time to have a "chance" of representing an interesting biological entity. This entity may correspond to a binding site, *i.e.* to a (in general small) part of a molecule that will interact with another, or it may represent an element that is repeated in a dispersed or periodic fashion (for instance, tandemly). The role played by a repetition of whatever type is often unknown: some repeats, in particular small tandem ones, have been implicated in a number of genetic diseases and are also interesting for the purposes of studying polymorphism; other types of repeats, such as short inverted ones, seem to be hotspots for recombination (roughly, the exchange of genetic material) intra and inter-species.

We start by discussing the notions of "non-identity", that is, of similarity, we shall be considering. We then introduce the two main types of pattern inference we shall address: *common pattern inference* and *repeat identification*. When the repetition of interest appears dispersed in a string, the problem is quite similar to identifying single patterns common to a set of strings. In fact, the second may be easily derived from the first. The case of tandem repeats, and of other forms of structured patterns, that is, of patterns composed of various parts at non-random distances from one another, requires a different treatment. Finally, we survey some algorithms for solving the various kinds of pattern inference problems under these different notions of similarity.

1.3 Notions of similarity

1.3.1 Preliminary definitions

If s is a string of length $|s| = n$ over an alphabet Σ, that is, $s \in \Sigma^n$, its individual elements (the letters composing it) will be denoted by s_i, $1 \leq i \leq n$. A nonempty word $u \in \Sigma^*$ in a string s is a factor $s_i s_{i+1} \ldots s_j$ for a given pair (i, j) such that $1 \leq i \leq j \leq n$. The empty word trivially is a factor of all strings. It is denoted by λ.

1.3.2 Identity

Although identity is seldom an appropriate notion of similarity to consider when working with biological objects, it may sometimes be of interest.

This is a straightforward notion we nevertheless define properly as this will allow us to introduce some notations that will be used throughout the paper.

The identity concerns words in a string and we therefore adopt Karp *et al.* [20] identification of such words by their start position in the string. To facilitate exposition, this and all other notions of similarity are given for words in a single string. It is straightforward to adapt them to the case of more than one string (for instance, by considering the string resulting from the concatenation of the initial ones with a distinct forbidden symbol separating any two adjacent strings). Let us denote by E the identity relation on the alphabet Σ (the letter E stands for "Equivalence").

Relation E between elements of Σ may then be extended to a relation E_k between words of length k in a string s in the following way.

Definition 1.3.1 *Given a string $s \in \Sigma^n$ and two positions i, j in s such that $i, j \leq n - k + 1$, then*

$$i \; E_k \; j \Leftrightarrow s_{i+l} \; E \; s_{j+l} \quad \textit{for all } l \textit{ such that } \; 0 \leq l \leq k - 1.$$

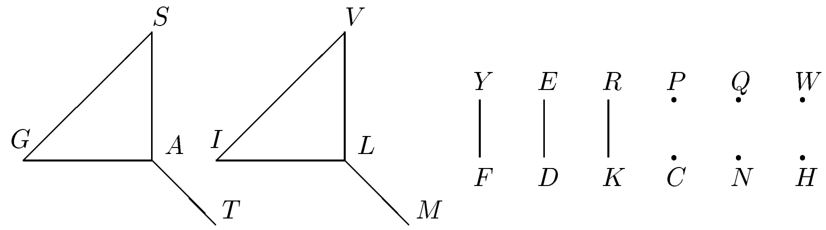
For each $k \geq 1$, E_k establishes an equivalence relation that corresponds to the identity relation between positions in a string s: two positions i and j are in the relation E_k if and only if the words of length k in s starting at positions i and j are identical. This gives us a first definition of similarity between such words. Indeed, each class of E_k of cardinality at least two represents a set of exactly repeated words in s.

1.3.3 Non-transitive relation

When dealing with biological strings one has to consider that the "letters" represented by such strings are complex biological objects with physico-chemical properties such as, for instance, electrical charge, polarity, size, different levels of acidity, etc. Some of these (but seldom all) properties may be shared by two or more of the objects. This applies more to proteins than to DNA/RNA but is true to some extent of both.

A more realistic relation to establish between the letters of the protein or DNA/RNA alphabet (respectively called amino acids and nucleotides) would therefore be reflexive, symmetric but non-transitive [40]. An example of such a relation, denoted by R, is given in Figure 1.1. It may be represented by a graph whose nodes are the elements of Σ, and where an edge links two nodes if the elements of Σ labeling the nodes correspond to biological objects sharing enough physico-chemical properties to be considered related.

Let $\Sigma = \{A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y\}$ be the alphabet of the amino acids and R be the relation of similarity between these amino acids given by the following graph:



The maximal cliques of R are the sets: $\{A, S, G\}$, $\{A, T\}$, $\{I, L, V\}$, $\{L, M\}$, $\{F, Y\}$, $\{D, E\}$, $\{K, R\}$, $\{C\}$, $\{P\}$, $\{N\}$, $\{Q\}$, $\{H\}$, $\{W\}$.

Figure 1.1. Example of a relation of similarity between the letters of the protein alphabet (called amino acids).

As in the previous section, a relation R between elements of Σ may easily be extended to a relation R_k between words of length k in a string s.

Definition 1.3.2 *Given a string $s \in \Sigma^n$ and two positions i, j in s such that $i, j \leq n - k + 1$, then*

$$i \ R_k \ j \Leftrightarrow s_{i+l} \ R \ s_{j+l} \ \text{ for all } l \text{ such that } \ 0 \leq l \leq k-1.$$

The concept that will be important to our setting will be that of a (maximal) clique of a non-transitive relation.

Definition 1.3.3 *Given an alphabet Σ and a non-transitive relation R on Σ, a set C of elements of Σ is a clique of relation R if $\alpha \ R \ \beta$ for all $\alpha, \beta \in C$. If C is a clique and $C \bigcup \{\gamma\}$ is not a clique for all $\gamma \in \Sigma \setminus C$, then C is called a maximal clique.*

Definition 1.3.4 *Given a string $s \in \Sigma^n$, a set C_k of positions in s is a clique of relation R_k if $i \ R_k \ j$ for all $i, j \in C_k$. If C_k is a clique and $C_k \bigcup \{l\}$ is not a clique for all $l \in [1..n] \setminus C_k$, then C_k is called a maximal clique of R_k.*

Maximal cliques of R_k give us then a second way of establishing a definition of similarity between words of length k in a string. If the similarity relation is transitive, the strings can be translated into a smaller alphabet and R is an identity relation E.

A parenthesis on protein structures

A non-transitive relation and its maximal cliques are a particularly appropriate tool for analyzing protein structures when such structures are coded as linear sequences of internal coordinates.

Indeed, the 3D structure of a protein is determined by the spatial arrangement of the atoms of its amino acids. The amino acids are linked together in a chain and a representation of the structure that preserves the linear order of the residues allows us to consider working with strings. When studying such structure, it is possible to focus attention on its backbone only. It is well known [4] that the local conformation of the backbone can be defined at each residue by three internal coordinates usually referred to as three dihedral angles: Φ, Ψ and ω [30]. For chemical reasons, ω is fixed (to 0 or 180°) and can usually be forgotten, and the internal coordinates of the backbone are therefore in general represented on a two-dimensional map Φ, Ψ called a Ramachandran map [30] (see Figure 1.2). The structure of the backbone can then be uniquely defined by the linear succession of the pairs of angles (Φ, Ψ) along the backbone (see Figure 1.3). Since these pairs of angles represent pairs of real values, they have to be recoded into discrete values so that we can work with an alphabet of discrete symbols. In order to do it, a grid of mesh $\epsilon°$ is constructed on the Ramachandran map [28, 37] (see again Figure 1.2) (note that 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. Each node of the lattice corresponds then to a symbol of a new alphabet. 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:

$$\forall(\alpha, \beta) \in \Sigma^2,\ \alpha\ R\ \beta \Leftrightarrow \exists \text{ a square of side } 2K\epsilon \text{ enclosing } \alpha \text{ and } \beta,$$

where α and β are nodes of the lattice and K is a parameter (called margin) that can be adjusted to broaden or narrow the matching precision. Reducing the values of ϵ and K, alphabets and relations of increasing sizes and degeneracies are produced (for instance, for a mesh of 5° the alphabet has 5182 symbols, and for $K = 1$ each symbol belongs to 9 distinct cliques of R).

In terms of angles, the previous definition simply means that two pairs of angles match if their corresponding symbols lie in a square of side $2K\epsilon$ of the Ramachandran map. The relation R is intrinsically non-transitive. The maximal cliques of R on Σ are, by definition, all (big) squares of side $K\epsilon$ centered on each node.

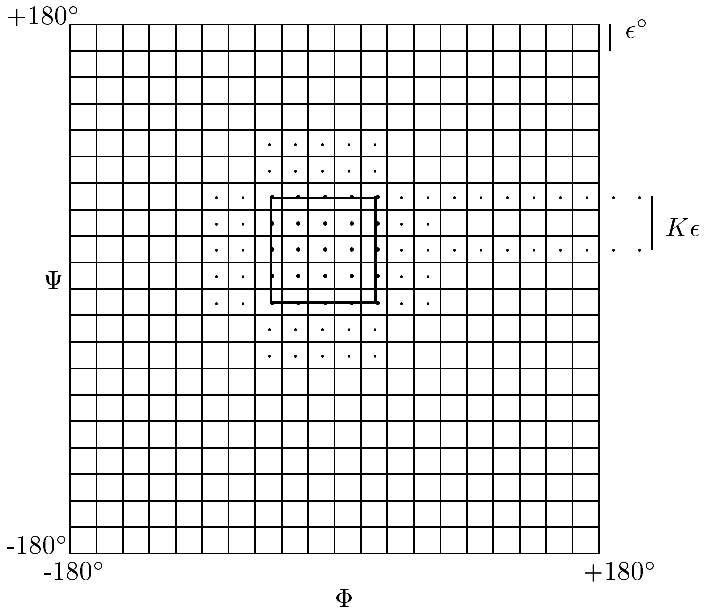


Figure 1.2. Sampled Ramachandran map: one square of side ϵ = one symbol of the alphabet.

What is important to note is that the maximal cliques of R_k represent then all structural contiguous motifs of length k of a set of protein structures.

A similar approach has also been developed to look for structural motifs in true three-dimensional space, that is, for motifs that are not necessarily composed of contiguous amino acids [9, 18].

1.3.4 Allowing for errors

1.3.4.1 Introducing the idea of a model

Let us initially assume the sole errors authorized are substitutions. In view of the definitions established in the previous sections, one would be tempted to define a relation of similarity H between two words of length k in a string s, that is, between two positions i and j in s, in the following way.

Definition 1.3.5 *Given a string $s \in \Sigma^n$ and two positions i, j in s such that $i, j \leq n - k + 1$, then*

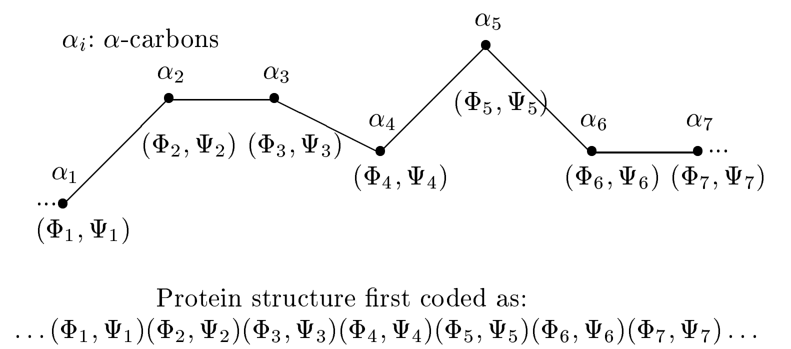


Figure 1.3. Structure of backbone defined as a linear succession of pairs of angles (Φ, Ψ).

$$i\ H_k\ j \Leftrightarrow dist_H(s_i \ldots s_{i+k-1}, s_j \ldots s_{j+k-1}) \leq e,$$

where $dist_H(u, v)$ is the Hamming distance between u and v (that is, the minimum number of substitutions to be performed on u to obtain v) and e is a nonnegative integer that is fixed.

Parameter e corresponds to the maximum number of substitutions one wishes to tolerate. In the same way as in Section 1.3.3, maximal cliques of H_k would provide us with another possible definition of similarity between words of length k in a string.

One could consider now how to adapt the above definition to the case of a Levenshtein distance L (which, in the case of strings, is the minimum number of substitutions, insertions and deletions necessary to obtain one string from another), or any other type of distance where insertions and deletions are permitted besides substitutions. This is not completely trivial: indeed, given two words u and v respectively starting at positions i and j in s and such that iL_kj, what is the meaning of k ? Before even trying, one may intuitively note that calculating H_k (and, *a fortiori*, L_k) is no longer as easy as calculating E_k or R_k.

The reason is that, although the definitions given in Sections 1.3.2 and 1.3.3 involve pairs of positions in a string s, it is possible to rewrite them in such a way that, given a position i in s and a length k (that of the words in s one is currently considering), it is immediate to determine to which class or clique(s) i belongs in the sense that the class or clique(s) can be uniquely identified just by "reading" $s_i \ldots s_{i+k-1}$. Let us consider first the simpler case of an identity. Straightforwardly, position i will belong to the class whose label is $s_i \ldots s_{i+k-1}$. In the case of a non-transitive relation R between the letters of Σ, let us name C the set of (maximal) cliques of R and denote by $clique_R(\alpha)$ the cliques of R to which a letter α belongs. Then, position i will belong to the sets of R_k whose labels may be spelled from the regular expression $clique_R(s_i) \dots clique_R(s_{i+k-1})$ and that are maximal under R_k. Note the small difference here with an identity relation: maximality of a validly labeled set has to be checked [40]; a class is always "maximal".

No such easy rewriting and verification are possible in the case of the definition of H_k (or L_k) if we wish to build the notion of similarity between words in a string upon that of the cliques of H_k. Indeed, obtaining such cliques requires comparing (a possibly large number of) pairs of positions between themselves. This is expensive.

One may, however, rewrite the definition of H_k in such a way that it refers to labels as we did above for E_k and R_k although such labels are no longer as immediately identifiable. A possible definition (still for the case where substitutions only are considered) would be the following.

Definition 1.3.6 *Given a string $s \in \Sigma^n$ and two positions i, j in s such that $i, j \leq n - k + 1$, then*

$$i \; H_k \; j \Leftrightarrow \exists m \in \Sigma^k \text{ such that } dist_H(m, s_i \dots s_{i+k-1}) \leq e \text{ and } dist_H(m, s_j \dots s_{j+k-1}) \leq e,$$

where $dist_H(u, v)$ and e are as before.

Generalizing this, we would then have:

Definition 1.3.7 *A set S_k of positions in s represents a set of words in s of length k that are all similar between themselves if, and only if, there exists (at least) one string $m \in \Sigma^k$ such that, for all elements i in S_k, $dist_H(m, s_i \dots s_{i+k-1}) \leq e$ and, for all $j \in [1..n] \setminus S_k$, $dist_H(m, s_j \dots s_{j+k-1}) > e$.*

Observe that extension of both definitions to a Levenshtein distance becomes now straightforward. We reproduce below, after modification, just the last definition.

Definition 1.3.8 *A set S_k of positions in s represents a set of words of length k that are similar if, and only if, there exists (at least) one string $m \in \Sigma^k$ such that, for all elements i in S_k, $dist_L(m, s_i \dots) \leq e$ and, for all $j \in [1..n] \setminus S_k$, $dist_L(m, s_j \dots) > e$.*

Since the length of the occurrence of a model m may now be different from that of m (it will vary between $|m| - e$ and $|m| + e$ where $|m|$ is the length of m), we denote it $(s_i \dots)$ leaving undefined its right-end point.

Observe also that it remains possible, given a position i in s and a length k, to obtain the label of the group(s) (we shall see that this is no longer a clique and is obviously not a class unless e is zero) of the relation H_k (or L_k) i belongs to. Such labels are represented by all strings $m \in \Sigma^k$ such that $dist_H(m, s_i \dots) \leq e$ (or $dist_L(m, s_i \dots) \leq e$), that is, such that their distance from the word starting at position i in s is no more than e.

We call *models* such group labels. Positions in s indicating the start of a word of length k are *e-occurrences* (or simply *occurrences* where there is no ambiguity) of a model m if $dist(m, s_i \ldots) \leq e$, where $dist$ is either the Hamming or Levenshtein distance. Observe that a model m may never be present exactly in s.

1.3.4.2 Models are not cliques, nor Steiner strings

When e is zero, models represent the classes of an identity relation. In the case where e is greater than zero, models such as defined in the previous section are neither classes nor cliques nor yet another well-defined mathematical object: Steiner strings. We recall that, given a distance $dist$ and a set of strings S, s is a *Steiner string* of the strings in S if $\sum_{u \in S} dist(s, u)$ is minimal.

Although Definition 1.3.6 could lead us to believe that models correspond to cliques, this is not the case. Consider $s \in \Sigma^* = \{\text{A, B, C}\}^*$ equal to ABACACABAAAA. If at most one substitution is allowed, then positions 1 and 5 (strings s are indexed from 1 to $|s|$) are related by H_4 because of models $m = ABAB$ and $m = ACAC$; so are positions 1 and 9 because of models $m = ABAA$ and $m = AAAC$ and, finally, positions 5 and 9 because of models $m = ACAA$ and $m = AAAB$. The set $\{1, 5, 9\}$ forms therefore a clique (any two elements are related by H_4), but it is never the same model for every possible pair. Indeed, there is no single model such that all three positions are occurrences of it.

The case of a Steiner string also calls for an example. Let us consider the string $s \in \Sigma^* = \{\text{A, B, C}\}^*$ equal this time to AAAAAABBBBACAC, and let us assume that at most one substitution is allowed. Consider $O = \{1, 2, 3, 11\}$ a set of positions in s. There exist only two models $m \in \Sigma^4$ such that $dist_H(m, s_i \ldots s_{i+3}) \leq e$ for all $i \in O$. These are models ACAA and AAAC. None is a Steiner string for O. The unique string that is a Steiner string for O is AAAA, which is not a model for O (it is at distance 2 from occurrence ACAC).

Nonetheless, models are interesting objects that provide a precise definition for the idea of conservation and will allow us, as we shall see later, to obtain reasonably efficient algorithms for identifying sets of conserved words in a string.

1.3.4.3 Expanding on the idea of models – Two more possible definitions of similarity

1.3.4.3.1 Non-transitive relation and errors

Models allow us to considerably enrich the notion of conservation. For instance, it enables us to simultaneously consider a non-transitive relation between the letters of the alphabet (amino acids or nucleotides) and the possibility of errors. In order to do that, it suffices to permit the model to be written over an extended alphabet composed of a subset of the set of all subsets of Σ (denoted $\mathcal{P}(\Sigma)$), where Σ is the alphabet of amino acids or nucleotides. Such an alphabet could be, for instance, one defined by the maximal cliques of the relation R given in Figure 1.1. Definition 1.3.8 of Section 1.3.4.1 then becomes:

Definition 1.3.9 *A set S_k of positions in s represents a set of words of length k that are all similar between themselves if, and only if, there exists (at least) one element $M \in P^k$ with $P \subseteq \mathcal{P}(\Sigma)$ such that, for all elements i in S_k, $setdist(M, s_i \ldots) \leq e$ and, for all $j \in [1..n] \setminus S_k$, $setdist(M, s_j \ldots) > e$, where $setdist(M, v)$ for $M \in P^*$ and $v \in \Sigma^*$ is the minimum Hamming or Levenshtein distance between v and all $u \in M$.*

Among the subsets allowed in P, the alphabet of models, may be $\{\Sigma\}$ itself, that is, the *wild card.* It is obvious that this may lead to trivial models. Alphabet P may then come with weights attached to each of its elements indicating how many times (possibly infinite) it may appear in a model of interest to us. Observe that another way of describing the alphabet P of models is as the set of edges of a (possibly weighted) hypergraph whose nodes are the elements of Σ.

When e is zero, we obtain a definition of similarity between words in the string that closely resembles that given in Section 1.3.3. Note however that, given two models M_1 and M_2, we may well have that the set of occurrences of M_1 is included in that of M_2. The cliques of Definition 1.3.4 will correspond to the sets of occurrences that are maximal.

1.3.4.3.2 A word instead of symbol-based similarity

Errors between a group of similar words and the model of which they are occurrences can either be counted as unitary events (possibly with different weights) as was done in the previous sections, or they can be given a score. The main idea behind scoring a resemblance between two objects is that it allows to average the differences that may exist between them. It may thus provide a more flexible function for measuring the similarity between words. A simple example illustrates this point in Figure 1.4.

In the example and in the definition of similarity introduced in this section, gaps are not allowed, only substitutions. This is done essentially for the sake of clarity. Gaps may, however, be authorized, the reader is referred to [38] for details.

Let a numerical matrix $\mathcal{M}$ of size $\mid \Sigma \mid \times \mid \Sigma \mid$ be given such that:

$$\mathcal{M}(a, b) = \text{score between } a \text{ and } b \text{ for all } a,\, b \in \Sigma.$$

If this score measures a similarity between a and b, we talk of a *similarity matrix* (two well-known examples of which in biology are PAM250 [7] and BLOSUM62 [14]), while if the score measures a dissimilarity between a and b we talk of a *dissimilarity matrix.* A special case of this latter matrix is

Let $\Sigma = \{A, B, C\}$ and
score $(\alpha, \alpha) = 1$, $\forall\ \alpha \in \Sigma$;
score $(A, B) =$ score $(B, A) = -1$;
score $(A, C) =$ score $(C, A) = -1$;
score $(B, C) =$ score $(C, B) = -1$.
If we say that two words are similar if either
- the number of substitutions between them is at most 1; or
- their score is at least 1;

then by the first criterion the words $AABAB$ and $AACCB$ are not similar, while by the second criterion they are, the second substitution being allowed because the two words on average share enough resemblance.

Figure 1.4. Example of the greater flexibility allowed by scoring words rather than counting errors.

when the dissimilarity measure is a metric, that is when the scores obey, among other conditions, the triangular inequality. In that situation, we talk of a *distance matrix* (an example of which is the matrix proposed by J.-L. Risler [31]).

In what follows, we consider $\mathcal{M}$ a similarity matrix.

Definition 1.3.10 *Given a string* $u = u_1 u_2 \ldots u_k \in \Sigma^k$, *a model* $m = m_1 m_2 \ldots m_k \in \Sigma^k$ *and a matrix* $\mathcal{M}$, *we define*

$$score_{\mathcal{M}}(u, m) = \sum_{i=1}^{k} \mathcal{M}(u_i, m_i).$$

Definition 1.3.11 *A set* S_k *of positions in* s *represents a set of words of length* k *that are similar if, and only if, given a positive integer* w *such that* $w \leq k$ *and a threshold value* t, *there exists (at least) one element* $m \in \Sigma^k$ *such that,*

1. *for all elements* i *in* S_k *and for all* $j \in \{1, \ldots, |m| - w + 1\}$, $score_{\mathcal{M}}(m_j \ldots m_{j+w-1}, s_{i+j-1} \ldots s_{(i+j-1)+(w-1)}) \geq t$;

2. *for all* $i \in [1..n] \setminus S_k$, *there exists at least one* $j \in \{1, \ldots, |m| - w + 1\}$ *such that* $score_{\mathcal{M}}(m_j \ldots m_{j+w-1}, s_{i+j-1} \ldots s_{(i+j-1)+(w-1)}) < t$.

An example is given in Figure 1.5.

Let $\Sigma = \{A, B, C\}$, $w = 3$ and $t = 6$. Let $\mathcal{M}$ be the following matrix:

	A	B	C
A	3	1	0
B	1	2	1
C	0	1	3

Given the three strings:

s1 = ABCBBABB$\underline{\text{BACABACB}}$BBAB
s2 = $\underline{\text{CABACAAC}}$BACCABCACCACCC
s3 = BBBACACCABA$\underline{\text{BACABACA}}$BA

then the longest model that is present in all strings is CACACACC (at positions 9, 1 and 12 respectively).

Figure 1.5. Example of a model under a word-based relation of similarity.

1.4 The problems and the properties associated with them

The previous sections presented various definitions of similarity between words in a string (easily extensible to words in a set of strings). The last definitions (from Section 1.3.4 on) introduced the notion of a model that was somewhat implicit in the earlier ones.

We discuss now further properties the models that will interest us must satisfy. Doing so, we also render such models more complex. To facilitate exposition, we henceforward denote models by a lower case m, whether m represents a word (*i.e.*, is defined over Σ) or a set of words (*i.e.*, is defined over $\mathcal{P}(\Sigma)$).

1.4.1 "Simple" models

Models such as given in the previous sections will be called *simple* models. There will be one important property such models (and, in some way or another, all other kinds of models) will have to satisfy and that is the following.

Property 1.4.1

Case of a single string. *Given a string s and a nonnegative integer q, a model m is said to be* valid *if it has at least q occurrences in s;*

Case of a set of strings. *Given a set of N strings and an integer q such that $1 \leq q \leq N$, a model m is said to be* valid *if it has at least one occurrence in at least q distinct strings of the set.*

Parameter q is called the *quorum* valid models must satisfy.

1.4.2 From simple to structured models

Although the objects defined in the previous section can be reasonable, algorithmically tractable models for single binding sites, they do not take into account the fact that such sites are often not alone (in the case of eukaryotes, they may even come in clusters) and, specially, that the relative positions of such sites when more than one participates in a biological process are in general not random. This is particularly true for some DNA binding sites such as those involved in the transcription of DNA into RNA (*e.g.* the so-called promoter sequences).

There is therefore a need for defining biological models as objects that take such characteristics into account. This has the motivation just mentioned but presents also interesting algorithmical aspects: exploiting such characteristics could lead to algorithms that are both more sensitive and more efficient. Models that incorporate such characteristics are called *structured models.*

1.4.2.1 Structured models

Formally, a *structured model* is a pair (m, d) where:

- m is a p-tuple of simple models $(m_1, \ldots, m_p)$ (representing the p parts a structured model is composed of – we shall call these parts *boxes*);
- d is a $(p-1)$-tuple of triplets $((d_{min_1}, d_{max_1}, \delta_1), \ldots, (d_{min_{p-1}}, d_{max_{p-1}}, \delta_{p-1}))$ (representing the $p-1$ intervals of distance between two successive boxes in the structured model);

with p a positive integer, $m_i \in \Sigma^+$ and d_{min_i}, d_{max_i} $(d_{max_i} \geq d_{min_i})$, δ_i nonnegative integers.

Given a set of N strings $s_1, \ldots, s_N$ and an integer $1 \leq q \leq N$, a model (m, d) is said to be *valid* if, for all $1 \leq i \leq p-1$ and for all occurrences u_i of m_i, there exist occurrences $u_1, \ldots, u_{i-1}, u_{i+1}, \ldots, u_p$ of $m_1, \ldots, m_{i-1}, m_{i+1}, \ldots, m_p$ such that:

- $u_1, \ldots, u_{i-1}, u_i, u_{i+1}, \ldots, u_p$ belong to the same string of the set;
- there exists d_i, with $d_{min_i} + \delta_i \leq d_i \leq d_{max_i} - \delta_i$, such that the distance between the end position of u_i and the start position of u_{i+1} in the string is equal to $d_i \pm \delta_i$;
- d_i is the same for p-tuples of occurrences present in at least q distinct strings.

The term d_i represents a distance and $\pm\delta_i$ an allowed interval around that distance. When $\delta_i = (d_{max_i} - d_{min_i} + 1)/2$, then δ_i is omitted and d in

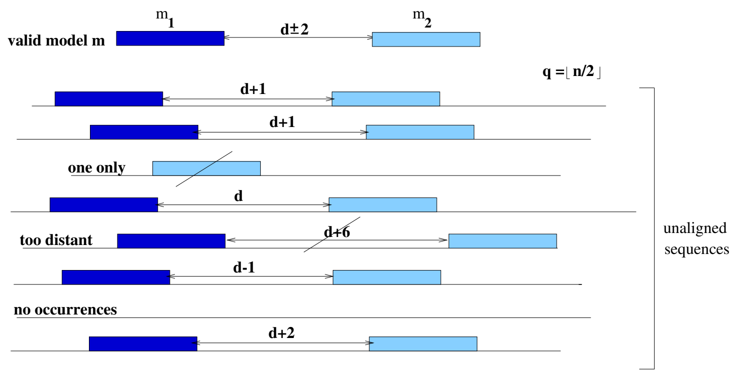


Figure 1.6. Example of a model with two boxes ($p = 2$).

a structured model (m, d) is denoted by a pair (d_{min_i}, d_{max_i}). An example of a model with $p = 2$ is given in Figure 1.6.

Observe that simple models are indeed but a special case of structured ones.

1.4.3 Models for tandem arrays (satellites)

Tandem arrays (called *tandem repeats* when there are only two units) are a sequence of repeats that appear adjacent in a string. In biology, such tandemly repeated units are divided into three categories depending on the length of the repeated element, the span of the repeat region and its location within the chromosome [2]. Repeats occurring in or near the centromeres and telomeres are called simply *satellites*. Their span is large, up to a million bases, and the length of the repeated element varies greatly, anywhere from 5 to a few hundreds of base pairs. In the remaining, euchromatic region, of the chromosome the kinds of tandem repeats found are classified as either *micro* or *mini satellites*, according to the length of the repeated element. Micro satellites are composed of short units, of 2 to 5 base pairs, in copy numbers in general around 100. Mini satellites on the other hand involve slightly longer repeats, typically around 15 base pairs, in clusters of variable sizes, comprising between 30 and 2000 elements.

Satellites of whatever type ask for a more complex definition of models that may initially recall that of the structured models presented in Section 1.4.2. Indeed, in the case of satellites, the models themselves are simple. However, although some constraints are imposed on the relative positions of the occurrences of a satellite model in a way that is somewhat similar to what was done with structured models, the overall nature of such constraints is not the same as will be developed in the next section.

1.4.3.1 Satellite models

We have in fact two definitions related to a satellite model, one called *prefix model* and the other *consensus model*. The latter will concern satellite models strictly speaking while prefix models are in fact models for approximately periodic repetitions that are not necessarily tandem.

Formally, a *prefix model* of a satellite is a string $m \in \Sigma^*$ (or $\mathcal{P}(\Sigma)$) that approximately matches a train of wagons. A *wagon* of m is a word u in s such that $dist(m, u) \leq e$. A *train* of a satellite model m is a collection of wagons $u_1, u_2, \ldots, u_p$ ordered by their starting positions in s and such that:

Property 1.4.2 *$p \geq min\_repeat$, where min_repeat is a fixed parameter that indicates the minimum number of elements a repeating region must contain.*

Property 1.4.3 *$left_{u_{i+1}} - left_{u_i} \in JUMP$, where $left_u$ is the position of the left-end of wagon u in s and*

$$JUMP = \{y : y \in \cup_{x \in [1, max\_jump]}\ x \times [min\_range, max\_range]\},$$

with the three parameters min_range, max_range and max_jump fixed.

A prefix model m is said to be *valid* if there is at least one train of m in the string s. Similarly, a train, when viewed simply as a sequence of substrings of s, is valid if it is the train for some model m. A prefix model represents the invariant that must be true as we progressively search for our final goal, which is to arrive at a *consensus model*. This is a prefix model which further satisfies the following:

Property 1.4.4 *$left_{u_{i+1}} - right_{u_i} \in GAP$, where $right_u$ is the position of the right-end of wagon u, and $GAP = \{y : y \in \cup_{x \in [0, max\_jump-1]}\ x \times [min\_range, max\_range]\}$.*

Parameter max_jump allows us to deal with very badly conserved elements inside a satellite (by actually not counting them) while we require that the satellite be relatively well conserved overall. Fixing max_jump at a value strictly greater than one means we allow some wagons (the badly conserved ones) to be "jumped". This may be seen as "meta-errors", that is as errors involving not a letter inside a wagon but a wagon inside a train. Note that $0 \in GAP$. This guarantees that, when jumps are not authorized, the repeats found are effectively tandem.

Since mutations affecting a unit concern *indels* (that is, insertions and deletions) as well as substitutions, it is sometimes interesting to work with a variant of the above properties where $JUMP$ and GAP are defined as

$$JUMP = \left\{ y : \begin{array}{l} y \in [min\_range, max\_range] \text{ or} \\ y \in \cup_{x \in [2, max\_jump]}\ x \times [min\_range - g, max\_range + g] \end{array} \right\}$$

$$GAP = \left\{ y : \begin{array}{l} y \in [min\_range, max\_range] \ or \\ y \in \cup_{x \in [1, max\_jump]} x \times [min\_range - g, max\_range + g] \end{array} \right\},$$

where $g \geq e$ is a fixed value. The idea is to allow the length of the badly conserved elements to vary more than is permitted for the detected "good" wagons.

1.5 Algorithms

We now develop the main ideas behind the algorithms for inferring patterns under the various definitions of similarity and properties specified in the previous sections. We focus on one algorithm each time, or a class of equivalent algorithms, the one that appears the most performing or flexible for our purposes (in general, the most easily adaptable to special cases or the introduction of additional constraints). Other methods are just mentioned. We also discuss (Section 1.5.3) the use of a special data structure for storing the string(s) itself (themselves).

We treat the case of one string only, except where considering more than one string increases the complexity of the algorithm in which case we address the multiple string problem instead. Once errors are permitted, we adopt the reference to models even though some of the earlier algorithms that handled errors did not explicitly make use of such external objects for extracting sets of similar words, or did not call them models.

Proofs are omitted and some details are skipped. Both may, in general, be found in the original papers. In all cases, we start by stating the problem that each specific algorithm intends to solve.

1.5.1 The simplest property: Identity

1.5.1.1 The simple identical repeated pattern problem

Perhaps the first classical algorithm for finding all exact repetitions in a string (henceforth called KMR) was elaborated by Karp, Miller and Rosenberg in 1972 [20]. Given a string s, KMR solves the following problems:

Problem 1.5.1 *Identify the positions of all words of a fixed length k that appear repeated in s.*

Problem 1.5.2 *Find the length k_{max} of the longest repeated word in s, and solve Problem 1.5.1 for $k = k_{max}$.*

1.5.1.2 Main ideas

KMR rests on the definition of the equivalence relation E_k given in Section 1.3.2. Problem 1.5.1 can then be formulated as the problem of finding the partition associated with E_k. Problem 1.5.2 requires finding the maximum value of k such that E_k is not the identity. The algorithm is based on an iterative construction of partitions E_l for $l \leq k$. The mechanism for performing such constructions rests on the following lemma.

Lemma 1.5.1 *Given two integers a, b with $1 \leq b \leq a$, and i, j two positions in a string s of length n, such that $i, j \leq n - (a + b) + 1$, then*

$$i \ E_{a+b} \ j \Leftrightarrow i \ E_a \ j \text{ and } (i + b) \ E_a \ (j + b).$$

The main idea behind the KMR algorithm is to use the lemma with $b = a$ for as long as possible. This means finding repeats of length $2a$ by using previouly acquired information on the repeats of length a that may become the prefixes and suffixes of those of length $2a$. If we are dealing with Problem 1.5.1, and if k is not a power of 2, we then use the lemma with $b < a$ in a last step in order to obtain E_k. If we are treating Problem 1.5.2, we may need more than one step to find the value of k_{max} such that $E_{k_{max}}$ is not the identity but $E_{k_{max+1}}$ is. The search for k_{max} from the smallest power of 2 that is bigger than k_{max}, let us say it is 2^p, can be done by applying the lemma with $b < a$ in a dichotomous fashion between 2^{p-1} and 2^p.

Constructing the partitions E_a basically corresponds to performing a set intersection operation. The intersections may be implemented using, for instance, stacks. More precisely, we need an array V_a of size n which stores, for each position i in s, the label of the class of E_a to which the a-long word starting at i belongs. The lemma is applied by means of two arrays of stacks P and Q. Stacks in P are filled by traversing V_a. Such stacks are in fact a dual of V_a. Each one corresponds to a class c of E_a and contains the positions i in s belonging to c. Array P serves therefore to sort the prefixes of length a of the repeats of length $2a$ one is trying to identify. The content of each stack of P in turn is then poured into the appropriate stack of Q. A division separates, within a same stack of Q, elements coming from different stacks of P. Like P, array Q has as many stacks as there are classes in E_a. It serves to sort the suffixes of length also a of the repeats of length $2a$. One then just needs to orderly pour Q into V_{2a} to obtain the classes of E_{2a}. In case the quorum is higher than 2, verifying that it is satisfied is a simple question of counting how many elements there are in each class.

Another method for obtaining the same result in a more efficient way that does not make use of complex data structures as will be discussed in Section 1.5.3 has been shown in [5]. It is, however, not easily extensible to the case of a non-transitive relation.

1.5.1.2.1 Complexity

Each partition construction takes $O(n)$ time, there are $O(\log k)$ such constructions, KMR time complexity is therefore $O(n \log k)$. When solving

Problem 1.5.2, this leads to an $O(n \log n)$ complexity because of possible degenerate cases (such as that of a string s composed of a single letter). KMR space complexity is $O(n)$.

1.5.2 More complex properties for single patterns: non-transitive relation without/with errors

1.5.2.1 Non-transitive relations without errors

1.5.2.1.1 The simple pattern problem under a non-transitive relation without error

KMR may be adapted to deal with a non-transitive relation R [40]. The problems solved will be the same as for KMR in the previous section.

1.5.2.1.2 Main ideas

Lemma 1.5.1 applies analogously, one just needs to substitute relation E by R.

Lemma 1.5.2 *Given two integers a, b with $1 \leq b \leq a$, and i, j two positions in a string s of length n, such that $i, j \leq n - (a + b) + 1$, then*

$$i \; R_{a+b} \; j \Leftrightarrow i \; R_a \; j \text{ and } (i + b) \; R_a \; (j + b).$$

Computing relations R_l for $l \leq k$ requires the same structures as for KMR, except that, as we saw, a set of positions pairwise-related by R_l is no longer a class but a clique. The algorithm was in consequence called KMRC (the "C" standing for Clique) [40]. In particular, a position may belong to two or more distinct cliques of R_l. Array V_l must therefore now be an array of stacks, like P and Q. It indicates, for each cell i corresponding to a position in s, the cliques of relation R_l to which i belongs.

The construction itself follows the same schema as indicated for KMR. Some of the sets of similar words obtained at the end of each step may not be maximal. A further operation is therefore needed to eliminate sets included in another one so as to get maximal cliques in the end.

1.5.2.1.3 Complexity

To calculate the complexity of the KMRC algorithm, we need to define a parameter g that measures the "degree of non-transitiveness" of relation R.

Definition 1.5.1 *Given R, a non-transitive relation on Σ, we call g the greatest number of cliques of R to which a symbol may belong, that is:*

$$g = Max \; \{g_a \mid a \in \Sigma, \; g_a = \text{number of cliques to which } a \text{ belongs}\}.$$

We call $\bar{g}$ the average value of g_a for $a \in \Sigma$, that is:

$$\bar{g} = \frac{\sum_a g_a}{n_c},$$

where n_c is the number of cliques of R.

If one does not count the set inclusion operations to eliminate non-maximal cliques, KMRC has time complexity $O(n \log k g^k)$ since each position i in s may belong to up to g^k (or, on average, $\bar{g}^k$) cliques of R_k. Inclusion tests based on comparing the positions contained in each set will take $O(n^2 g^{2k})$ time at the end of step k. At least another approach for testing set inclusion is possible and may result in a better theoretical time complexity (but not necessarily better in practice – this is discussed in [40]). Space complexity is $O(ng^k)$.

Another combinatorial approach for solving the problems addressed by KMRC uses an idea that is quite close to that of inferring models and will be discussed in the next section.

1.5.2.2 Non-transitive relations with errors

1.5.2.2.1 The simple pattern problem with errors

Models are considered this time. The problem we wish to solve is then the following.

Problem 1.5.3 *Given a string s, an integer $e \geq 0$ and a quorum q, find all models m such that m is valid, that is, is present at least q times in s, each time with at most e errors.*

1.5.2.2.2 Main ideas

KMR's principle for finding all exact repetitions of length $2k$ in a string s is based on the idea that such repetitions can be decomposed into two adjacent repetitions of length k.

In a likewise manner, we could base the principle of an algorithm for finding all the models of length k that are present with errors in a string s on an iterative construction that would double the length of the models at each step. Indeed, the set of occurrences of a model $m = m_1 m_2$ with $|m| = 2 \times |m_1| = 2 \times |m_2| = 2k$ is the set of occurrences of m_1 which are juxtaposed with at least one occurrence of m_2. As for KMR, we need to obtain and stock the sets of occurrences of all the valid models of length k in order to obtain those of length $2k$.

If we consider the search space of the problem, which corresponds to the tree of all possible models (in fact a trie), KMR, KMRC and the algorithm we are proposing now thus perform a breadth-first exploration of such search tree. All levels are not visited since at each step a lot of pruning may be realized (cutting off whole subtrees whose root is not labeled by a valid model) but all information concerning a given level, say k, is needed to build a deeper level, say $2k$. If errors are allowed, a breadth-first exploration of the tree may therefore consume a lot of memory in the earlier stages, when almost all models occur at almost all positions.

A second, more space-parsimonious approach to constructing such models is to traverse the tree depth-first, again with possible pruning along the way. Indeed, if instead of doubling the length of the models at each step, we extend each model separately to the right by just one unit at a time, then all we need to obtain the set of occurrences of a model m is the set of occurrences of the model m' of length $|m|$ - 1 that is its prefix, plus a look at what follows each such occurrence in the string. In terms of memory, all we need to stock at any time is therefore the sets of occurrences of all the models that are prefixes of the currently considered model m.

The lemma that is applied is the following. Observe that, to facilitate the "look at what follows in the string", occurrences are now identified by their right-end positions instead of left-ends. They come also accompanied by the number of errors that they have accumulated against the model. They are therefore represented by a pair (i, d) where i is a position in s and d is a distance.

Lemma 1.5.3 *Pair (i, d) is an occurrence of model $m\alpha$ with $\alpha \in \Sigma$ if, and only if, $d \leq e$ and at least one of the following is true:*

- **(match)** $(i-1, d)$ *is an occurrence of m and $s_i = \alpha$;*
- **(substitution)** $(i-1, d-1)$ *is an occurrence of m and $s_i \neq \alpha$;*
- **(deletion)** $(i, d-1)$ *is an occurrence of m;*
- **(insertion)** $(i-1, d-1)$ *is an occurrence of $m\alpha$.*

Note that applying the lemma corresponds to doing sparse dynamic programming between string s and the virtual trie $\mathcal{M}$ of all possible valid models. It is sparse because we are looking for models having occurrences at a maximum distance and therefore only a few cells of the usual dynamic programming matrix need to be kept.

Other, earlier approaches had been elaborated to infer patterns with errors. They implied generating all possible words of a given length and identifying their occurrences by simple pattern matching against the string [12, 29, 41, 47, 48, 49].

Extension of the algorithm to deal with models m defined over $\mathcal{P}(\Sigma)^*$ is straighforward [17, 16] (no error) [39]. There is, however, a notion of redundancy that appears at the level of models over $\mathcal{P}(\Sigma)^*$ that is not trivial to treat. The interested reader is referred to [27] and [36] for further details.

1.5.2.2.3 Complexity

In the case of no error, the difference in time complexity between KMR/KMRC and the algorithm sketched above varies only in that a $\log k$ term for the first approach is changed into a k term for the last one.

Before giving the complexity for the case where errors are permitted, we need to introduce the notion of a *word neighbourhood*.

Definition 1.5.2 *Given a word of length k defined over Σ^k, its e-neighbourhood, denoted $\mathcal{V}(e,k)$, is the number of words situated at a distance (Hamming or Levenshtein) at most e from it.*

In [33] and [39], we show that $\mathcal{V}(k,e)$ is bounded above in both cases by $k^e|\Sigma|^e$.

The neighbourhood measures the number of models $m \in \Sigma^k$ of which a position i in s may be an occurrence. In the case where models are in $\mathcal{P}(\Sigma)^k$ instead, $k^e|\Sigma|^e$ must be further multiplied by g^k. Since there are $O(n)$ positions, and the algorithm takes k steps, the time complexity of the algorithm sketched in Section 1.5.2.2.2 is $O(nk\mathcal{V}(e,k))$ or $O(nkg^k\mathcal{V}(e,k))$. The space complexity is $O(n)$ or $O(ng^k)$.

It is worth pointing out that, when e is zero, we obtain algorithms that have time complexity $O(nk)$ and $O(nkg^k)$ respectively. As we saw, KMR and KMRC obtain better theoretical performances. Furthemore, at least for models defined over Σ and an identity relation, it is possible to have even better results, in particular but not exclusively, by making use of special data structures such as suffix trees. The reader will find extensive discussion of such use for the "no error" case in [13]. We discuss below the use of suffix trees for inferring models with errors. Since considering more than one input string makes a difference to the algorithm in terms of time and space, we address the case of multiple input strings. Deducing equivalent algorithms and their complexities for the case of just one string is straighforward and will not be detailed.

1.5.3 Introducing suffix trees

The idea behind using suffix trees, as will be developed below, comes from the observation that long strings, specially when they are defined over a small alphabet, may contain many exact repetitions. One does not want to compare such repeated parts more than once with the potentially valid models. One way of doing that is using a representation of the string s that allows to put together some of the repetitions, that is, using an index of s such as a suffix tree $\mathcal{T}$.

1.5.3.1 Building the suffix tree

We do not describe the suffix tree construction, this can be found in either [25, 42] or (for a review of this and other data structures and text algorithms) [6] and [13]. We just recall some of the basic properties such structures possess (these are taken from [25]).

Basic properties of the suffix tree $\mathcal{T}$ of a string s

Property 1.5.1 *An arc of $\mathcal{T}$ may represent any nonempty substring of s.*

Property 1.5.2 *Each node of $\mathcal{T}$ that is not a leaf, except for the root, must have at least two offspring arcs (compact version of the tree).*

Property 1.5.3 *The strings represented by sibling arcs of $\mathcal{T}$ must begin with different symbols of Σ.*

Observe that Property 1.5.2 means an arc of $\mathcal{T}$ may be labeled by an element of Σ^k for $k \geq 2$ (for space considerations, each arc of $\mathcal{T}$ is in fact labeled by a pair of numbers corresponding to the start and end positions in s of the substring it represents, or its start position and length). Furthermore, an edge links every node spelling αx with $\alpha \in \Sigma$ and $x \in \Sigma^*$ to the node spelling x. Such edges are called *suffix links* and are what allows the tree to be built in time linear with the length of the string.

The key feature of a suffix tree is that for any leaf i, the concatenation of the labels of the arcs on the path from the root to i spells the suffix of s starting at position i. Reciprocally, the path spelled by every suffix of s leads to a distinct leaf if we assume that the last symbol of s appears nowhere else in s. To achieve this, we just need to concatenate at the end of s a symbol not appearing in Σ.

1.5.3.2 Generalized suffix trees

Trees for representing all the suffixes of a set of strings $\{s_i,\ 1 \leq i \leq N$ for some $N \geq 2\}$ are called *generalized suffix trees* and are constructed in a way very similar to the construction of the suffix tree for a single string [1, 15]. We denote such generalized trees by $\mathcal{GT}$. They share all the properties of a suffix tree given in Section 1.5.3.1 with, in Property 1.5.1, string s substituted by strings $s_1, \ldots, s_N$.

In particular, a generalized suffix tree $\mathcal{GT}$ verifies the fact that every suffix of every string s_i in the set leads to a distinct leaf. When $p \geq 2$ strings have a same suffix, the generalized tree has therefore p leaves corresponding to this suffix, each associated with a different string. To achieve this property during construction, we just need to concatenate to each string s_i of the set a symbol that is not in Σ and is specific to that string.

1.5.3.3 Adding information to the nodes of the tree

To be able to spell valid models (*i.e.* models satisfying the quorum constraint), we need to add some information to the nodes of the suffix tree.

In the case where we are looking for repeats in a single string s, we just need to know, for each node x of $\mathcal{T}$, how many leaves are contained in the subtree rooted at x. Let us denote $leaves_x$ this number for each node x. Such information can be added to the tree by a simple traversal of it.

If we are dealing with $N \geq 2$ strings, and therefore a generalized suffix tree $\mathcal{GT}$, it is not enough anymore to know the value of $leaves_x$ for each node x in $\mathcal{GT}$ in order to be able to check whether a model remains valid.

Indeed, for each node x, we need this time to know not the number of leaves in the subtree of $\mathcal{GT}$ having x as root, but that number for each different string the leaves refer to.

In order to do that, we must associate to each node x in $\mathcal{GT}$ an array, denoted $colours_x$, of dimension N, that is defined by:

$$colours_x[i] = \begin{cases} 1, & \text{if at least one leaf in the subtree rooted at } x \\ & \text{represents a suffix of } s_i; \\ 0, & \text{otherwise,} \end{cases}$$

for $1 \leq i \leq N$.

The array $colours_x$ for all x may also be obtained by a simple traversal of the tree with each visit to a node taking $O(N)$ time. The additional space required is $O(N)$ per node.

1.5.3.4 Spelling the models

The main difference with the approach described in sections 1.5.1, 1.5.2.1 and 1.5.2.2 is that occurrences are now grouped into classes and "real" ones, that is occurrences considered as individual words in the strings, are never directly manipulated. Present case occurrences of a model are thus in fact nodes of the suffix tree (we denote them by the term "node-occurrences") and are extended *in* the tree instead of in the string. Once the process of model spelling has ended, the start positions of the "real" occurrences of the valid models may be recovered by traversing the subtrees of the nodes reached so far and reading the labels of their leaves.

The algorithm is a development of the recurrence formula given in the lemma below where x denotes a node of the tree, $father(x)$ its father and d the number of errors between the label of the path going from the root to x as against a model m.

Lemma 1.5.4 *(x, d) is a node-occurrence of $m' = m\alpha$ with $m \in \Sigma^k$ and $\alpha \in \Sigma$ if, and only if, $d \leq e$ and one of the following two conditions is verified:*

(match)	*$(father(x), d)$ is a node-occurrence of m and the label of the arc from $father(x)$ to x is α;*
(substitution)	*$(father(x), d-1)$ is a node-occurrence of m and the label of the arc from $father(x)$ to x is $\beta \neq \alpha$;*
(deletion)	*$(x, d-1)$ is a node-occurrence of m;*
(insertion)	*$(father(x), d-1)$ is a node-occurrence of $m\alpha$.*

1.5.3.5 Complexity

If n is the average length of the strings and N their number, creating $colors_x$ for each node x of the tree takes time $O(nN^2)$, however manipulating it requires $O(N)$ time per model. Since there can be $O(nN\mathcal{V}(e,k))$ valid models in the worst case, the algorithm time complexity becomes $O(nN^2\mathcal{V}(e,k))$.

1.5.4 Structured models

1.5.4.1 The structured model problems

Concerning structured models, solutions to variants of increasing generality of a same basic problem are proposed. Suffix trees are used in all cases. These variants may be stated as follows. Given a set of N strings $s_1, \ldots, s_N$, a nonnegative integer e and a positive integer q.

Problem 1.5.4 *Find all models* $((m_1, m_2), (d_{min_1}, d_{max_1}))$ *that are valid.*

Problem 1.5.5 *Find all models* $((m_1, \ldots, m_p),\ ((d_{min_1},\ d_{max_1}),\ \ldots,\ (d_{min_{p-1}},\ d_{max_{p-1}})))$ *that are valid, where* $p \geq 2$.

Problem 1.5.6 *Find all models* $((m_1, m_2), (d_{min_1}, d_{max_1}, \delta_1))$ *that are valid.*

Problem 1.5.7 *Find all models* $((m_1, \ldots, m_p),\ ((d_{min_1}, d_{max_1}, \delta_1), \ldots, (d_{min_{p-1}}, d_{max_{p-1}}, \delta_{p-1})))$ *that are valid, where* $p \geq 2$.

The last two problems represent situations where the exact intervals of distances separating the parts of a structured site are unknown, the only known fact being that these intervals cover a restricted range of values. How restricted is indicated by the δ_i parameters.

To simplify matters, we shall consider that, for $1 \leq i \leq p$, $m_i \in \Sigma^k$ where k is a positive integer, *i.e.* each single model m_i of a structured model (m, d) is of fixed, unique length k. In a likewise manner, we shall assume that each part m_i has the same substitution rate e and, when dealing with models composed of more than two boxes, that the d_{min_i}, d_{max_i} and, possibly, δ_i for $1 \leq i \leq p-1$ have identical values. We denote by d_{min}, d_{max} and δ these values. Problem 1.5.5 is then formulated as finding all models $((m_1, \ldots, m_p), (d_{min}, d_{max}))$ that are valid and Problem 1.5.7 as finding all valid models $((m_1, \ldots, m_p), (d_{min}, d_{max}, \delta))$.

Besides fixing a maximum substitution rate for each part in a structured model, one can also establish a maximum substitution rate for the whole model. Such a global error rate allows to consider in a limited way possible correlations between boxes in a model.

Another possible global, or local, constraint one may wish to consider for some applications concerns the composition of the boxes. One may, for instance, determine that the frequency of one or more nucleotide in a box (or among all boxes) be below or above a certain threshold. For structured models composed of more than p boxes, one may also establish that a box i is palindromic in relation to a box j for $1 \leq i < j \leq p$. In algorithmical terms, the two types of constraints just mentioned are not equivalent. The first type, box composition whether local or global, can in general be verified only *a posteriori* while the second type (palindromic boxes) will result in a, sometimes substantial, pruning of the virtual trie of models.

Introducing such additional constraints may in some cases ask for changes to the basic algorithms described below. The interested reader may find the details concerning such changes in the original paper [23, 24].

1.5.4.2 Algorithms for a known interval of distance (Problem 1.5.4)

We present first a naive approach then two algorithms that are efficient enough to tackle structured model extraction from big datasets. The second algorithm has a better time complexity than the first but needs more space. The first is easier to understand and implement. Both are described in more detail than previous algorithms as structured models in some ways incorporate almost all other kinds of patterns we have been considering. The most notable exception concerns satellites that will be discussed in Section 1.5.5.

Other combinatorial approaches were developed for treating somewhat similar kinds of structured patterns. They either enumerate all possible (not just valid) patterns [43], do not allow for errors [17, 16] or are heuristics [11, 21].

1.5.4.2.1 Naive approach

A naive way of solving Problem 1.5.4 consists in extracting and storing all valid single models of length k (given q and e), and then, once this is finished, in checking which pairs of such models could represent valid structured models (given an interval of distance $[d_{min}, d_{max}]$).

The lemma used for building valid single models is the same as in Section 1.5.3 except that in practice, for most biological problems we wish to address [44, 45], substitutions only will in general be allowed. The lemma therefore becomes:

Lemma 1.5.5 *(x, d) is a node-occurrence of $m' = m\alpha$ with $m \in \Sigma^k$ and $\alpha \in \Sigma$ if, and only if, $d \leq e$ and one of the following two conditions is verified:*

(match)	*$(father(x), d)$ is a node-occurrence of m and the label of the arc from $father(x)$ to x is α;*
(substitution)	*$(father(x), d-1)$ is a node-occurrence of m and the label of the arc from $father(x)$ to x is $\beta \neq \alpha$.*

One way of doing the verification profits from the simple observation that two single models m_1 and m_2 may form a structured one if, and only if, at least one occurrence of m_1 is at the right distance of at least one occurrence of m_2. Building an array of size nN where cell i contains the list of models having an occurrence starting at that position in $s = s_1 \dots s_N$ allows to compare models in cell i to models in cells $i + d_{min}, \dots, i + d_{max}$ only. If the sets of occurrences of models are ordered, this comparison may be done in an efficient way (in time proportional to the size of the sets of node-occurrences, which is upper-bounded by nN).

1.5.4.2.2 First algorithm: Jumping in the suffix tree

A first non-naive approach to solving the problem starts by extracting single models of length k. Since we are traversing the trie of models in depth-first fashion (also in lexicographic order), models are recursively extracted one by one. At any time, a single model m (and its prefixes) is being considered. Once a valid model m_1 of length k is obtained together with its set of $\mathcal{T}$-node-occurrences V_1 (which are nodes located at level k in $\mathcal{GT}$), the extraction of all single models m_2 with which m_1 could form a structured model $((m_1, m_2), (d_{min}, d_{max}))$ starts. This is done with m_2 representing the empty word and having as node-occurrences the set V_2 given by:

$$\begin{aligned} V_2 \quad = \quad & \{(w, e_w = e_v) \mid \text{there exists } v \text{ in } V_1 \text{ ancestor of } w \text{ with} \\ & d_{min} \leq \ level(w) - \ level(v) \leq d_{max}\}, \end{aligned}$$

where level(v) indicates the level of node v in $\mathcal{GT}$. From a node-occurrence v in V_1, a jump is therefore made in $\mathcal{GT}$ to all potential start node-occurrences w of m_2. These nodes are the d_{min}- to d_{max}-generation descendants of v in $\mathcal{GT}$. Exactly the same recurrence formula given in Lemma 1.5.5 may be applied to the nodes w in V_2 to extract all single models m_2 that, together with m_1 could form a structured model verifying the conditions of the problem, for all valid m_1. An illustration is given in Figure 1.7 and a pseudo-code is presented below. The procedure ExtractModels is called with m the empty word having as sole node-occurrence the root of $\mathcal{GT}$ and $i = 1$.

procedure ExtractModels(Model m, Block i)
1. **for** each node-occurrence v of m **do**
2. **if** $i = 2$ **then**
3. put in *PotentialStarts* the children w of v at levels $k + d_{min}$ to $k + d_{max}$
4. **else**
5. put v (*i.e.*, the root) in *PotentialStarts*
6. **for** each model m_i (and its occurrences) obtained by doing a recursive depth-first traversal from the root of the virtual model tree $\mathcal{M}$ while $\mathcal{M}$ while simultaneously traversing $\mathcal{GT}$ from the node-occurrences in *PotentialStarts* (Lemma 1.5.5 and quorum constraint) **do**
7. **if** $i = 1$ **then**
8. ExtractModels($m = m_1$, $i + 1$)
9. **else**
10. report the complete model $m = ((m_1, m_2), (d_{min}, d_{max}))$ as valid

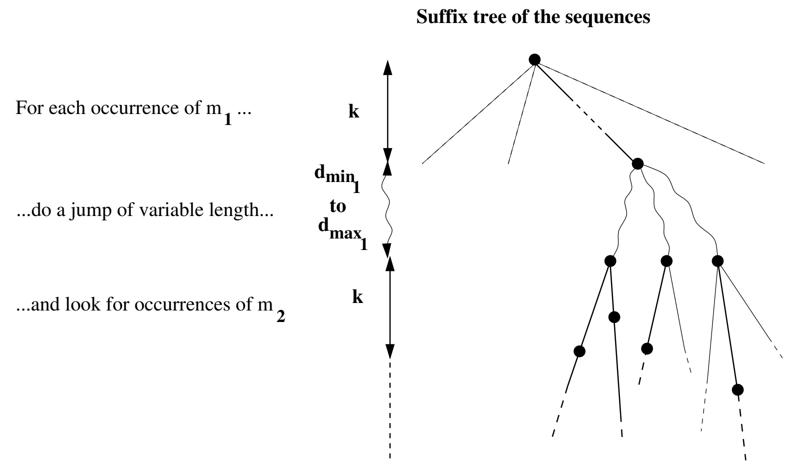


Figure 1.7. Extracting structured models (in the context of Problem 1) with a suffix tree – An illustration of Algorithm 1.

Since the minimum and maximum length of a structured model (m, d) that may be considered are, respectively, $2k + d_{min}$ and $2k + d_{max}$, we need only build the tree of suffixes of length $2k + d_{min}$ or more, and for each such suffix to consider at most the first $2k + d_{max}$ symbols.

The observation made in the previous paragraph applies also to the second algorithm (Section 1.5.4.2.3 below). Note that, in both cases, this implies $n_i \leq n_{i+1} \leq Nn$ for all $i \geq 1$ where n_i is the number of nodes at depth i in $\mathcal{GT}$.

1.5.4.2.3 Second algorithm: Modifying the suffix tree

The second algorithm initially proceeds like the first: it starts by building single models of length k, one at a time. For each node-occurrence v of a first part m_1 considered in turn, a jump is made in $\mathcal{GT}$ down to the descendants of v situated at lower levels. This time however, the algorithm just passes through the nodes at these lower levels, grabs some information the nodes contain and jumps back up to level k again (in a way that will be explained in a short while). The information grabbed in passing is used to temporarily and partially modify $\mathcal{GT}$ and start, *from the root of* $\mathcal{GT}$, the extraction of the second part m_2 of a potentially valid structured model $((m_1, m_2), (d_{min}, d_{max}))$. Once the operation of extracting all possible companions m_2 for m_1 has ended, that part of $\mathcal{GT}$ that was modified is restored to its previous state. The construction of another single model m_1 of a structured model $((m_1, m_2), (d_{min}, d_{max}))$ then follows, and the whole process unwinds in a recursive way until all structured models satisfying the initial conditions are extracted.

More precisely, the operation between the spelling of models m_1 and m_2 locally alterates $\mathcal{GT}$ up to level k to a tree $\mathcal{GT}'$ that contains only the k-long prefixes of suffixes of $\{s_1, \ldots, s_N\}$ starting at a position between d_{min} and d_{max} from the end position in s_i of an occurrence of m_1. Tree $\mathcal{GT}'$ is, in a sense, the union of all the subtrees t of depth at most k rooted at nodes that represent start occurrences of a potential companion m_2 for m_1.

For each model m_1 obtained, before spelling all possible companions m_2 for m_1, the content of $colors_z$ for all nodes z at level k in $\mathcal{GT}$ are stored in an array L of dimension n_k (this is for later restoration of $\mathcal{GT}$). Tree $\mathcal{GT}'$ is then obtained from $\mathcal{GT}$ by considering all nodes w in $\mathcal{GT}$ that may be reached on a descent of, this time, $k + d_{min}$ to $k + d_{max}$ arcs down from the node-occurrences (v, e_v) of m_1. These correspond to all end node-occurrences (instead of start as in the first algorithm) of potentially valid models having m_1 as first part. The boolean arrays $colors_w$ for all w indicate to which input strings these occurrences belong. This is the information we grab in passing and take along the only path of suffix links in $\mathcal{GT}$ that leads back to a node z at level k in $\mathcal{GT}$. If it is the first time z is reached, $colors_z$ is set equal to $colors_w$, otherwise $colors_w$ is added (boolean "or" operation) to $colors_z$. Once all nodes v and w have been treated, the information contained in the nodes z that were reached during this operation are propagated up the tree from level k to the root (using normal tree arcs) in the following way: if $\bar{z}$ and $\hat{z}$ have same parent z, then $colors_z = colors_{\bar{z}} \cup colors_{\hat{z}}$. Any arc from the root that is not visited at least once in such a traversal up the tree is not part of $\mathcal{GT}'$, nor are the subtrees rooted at its end node.

The extraction of all second parts m_2 of a structured model (m, d) follows as for single models in the initial algorithm (Lemma 1.5.5 in Section 1.5.4.2.1).

Restoring the tree $\mathcal{GT}$ as it was before the operations described above requires restoring the value of $colors_z$ preserved in L for all nodes z at level k and propagating the information (state of boolean arrays) from z up to the root.

Since nodes w at level between $2k + d_{min}$ to $2k + d_{max}$ will be solicited for the same operation over and over again, which consists in following the unique suffix-link path from w to a node z at level k in $\mathcal{GT}$, $\mathcal{GT}$ is pre-treated so that one single link has to be followed from z. Going from w to z takes then constant time.

An illustration is given in Figure 1.8. A pseudo-code for the algorithm is as follows. The procedure ExtractModels is called, as for the first algorithm, with m the empty word having as sole node-occurrence the root of $\mathcal{GT}$ and with i equal to 1.

procedure ExtractModels(Model m, Block i)
for each node-occurrence v of m **do**
    if $i = 2$ **then**
        put in *PotentialEnds* the children w at levels $2k + d_{min}$ to $2k + d_{max}$
        for each node-occurrence w in *PotentialEnds* **do**
            follow fast suffix-link to node z at level k
            put z in L
            if first time z is reached **then**
                initialize $colors_z$ with zero
                put z in *NextEnds*
            add $colors_w$ to $colors_z$
        do a depth-first traversal of $\mathcal{GT}$ to update the boolean arrays from the root to all z in *NextEnds* (let $\mathcal{GT}'$ be the k-deep tree obtained by such an operation)
    if $i = 1$ **then**
        Tree $= \mathcal{GT}$
    else
        Tree $= \mathcal{GT}'$
for each model m_i (and its occurrences) obtained by doing a recursive depth-first traversal from the root of the virtual model tree $\mathcal{M}$ while simultaneously traversing *Tree* from the root (Lemma 1.5.5 and quorum constraint) **do**
    if $i = 1$ **then**
        ExtractModels($m = m_1$, $i + 1$)
    else
        report the complete model $m = ((m_1, m_2), (d_{min}, d_{max}))$ as valid
        restore tree $\mathcal{GT}$ to its original state using L

Proposition 1.5.1 *The following two statements are true:*

- *$\mathcal{GT}'$ contains only the k-long prefixes of suffixes of $\{s_1, \ldots, s_N\}$ that start at a position between d_{min} and d_{max} of the end position in $\{s_1, \ldots, s_N\}$ of an occurrence of m_1;*
- *the above algorithm solves Problem 1.5.4.*

The proof is straightforward and may be found in [23, 24].

1.5.4.3 Complexity

The naive approach to solving Problem 1.5.4 requires $nN^2\ \mathcal{V}(e,k)$ time to find single models that could correspond to either part of a structured model (and $nN\mathcal{V}(e,k)$ space to store all potential parts). If we denote by Δ the value $d_{max} - d_{min} + 1$, finding which pair of single models may be

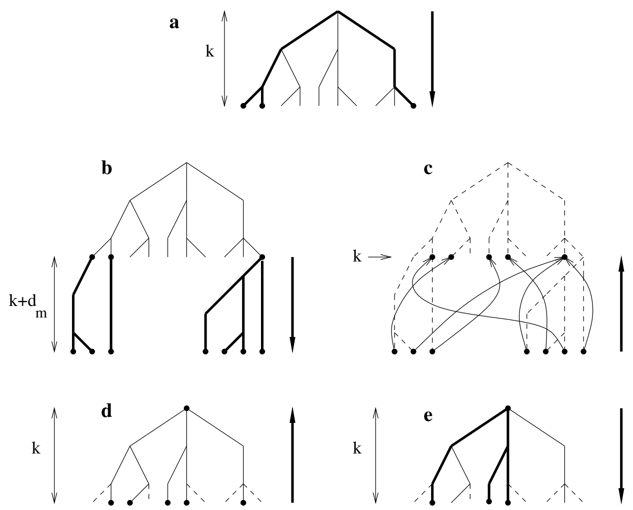


Figure 1.8. Extracting structured models (in the context of Problem 1) with a suffix tree – An illustration of Algorithm 2.

put together to produce a structured model could then be done in time proportional to:

$$\underbrace{\mathcal{V}(e,k)}_{(1)}\underbrace{\Delta\mathcal{V}(e,k)}_{(2)}\underbrace{nN}_{(3)}\underbrace{nN}_{(4)}$$

where (1) is the maximum number of single models to which a position may belong, (2) is the maximum number of models to which a position at a distance between $k+d_{min}$ and $k+d_{max}$ from the first may belong, (3) is the maximum number of comparisons that must be done to check whether two single models may form a structured one and, finally, (4) is the number of starting positions to consider.

To obtain the complexity of the first algorithm, we have to calculate the total number of visits we may do to nodes between level $2k+d_{max}$ (the deeper level we ever reach) and the root. To count this, we need to consider, for each node between levels $2k+d_{min}$ and $2k+d_{max}$ in $\mathcal{GT}$, how many times it could represent the node-occurrence of a model composed of two boxes, each one having length k and separated by a spacer of length d_{min} to d_{max}. This number is at most:

$$\begin{array}{rcl} \sum_{i=d_{min}}^{d_{max}} n_{2k+i}\mathcal{V}^2(e,k) & \leq & \min\{2,\Delta\}n_{2k+d_{max}}\mathcal{V}^2(e,k) \\ & \leq & \min\{2,\Delta\}n_{2k+d_{max}}k^{2e}|\Sigma|^{2e}, \end{array}$$

where Δ denotes the value $d_{max}-d_{min}+1$ and $n_{2k+d_{max}}$ is the number of tree nodes at depth $2k+d_{max}$. This last number is never more than nN. The $\min\{2, \Delta\}$ in the bound comes from the fact that the degree of any internal node of $\mathcal{GT}$ is at least 2. Since each visit to a node requires at most $O(N)$ operations, the time complexity of the first algorithm is $O(\min\{2, \Delta\} N n_{2k+d_{max}} \mathcal{V}^2(e, k))$, that is, $O(N n_{2k+d_{max}} \mathcal{V}^2(e, k))$. The space complexity is $O(N^2 n)$, as for the extraction of single models.

In the case of the second algorithm, we need to consider the number of operations necessary for building the two parts of each model using $\mathcal{GT}$ or $\mathcal{GT}'$, as well as the number of operations needed to obtain $\mathcal{GT}'$ from $\mathcal{GT}$ and then to restore back $\mathcal{GT}$.

The single models composing either two parts of a structured model may be built in at most $N n_k \mathcal{V}^2(e, k)$ operations. The reason for this is that, when spelling either part of a model, we are working with nodes between the root and level k only (there are at most $2n_k$ such nodes), and there are $\mathcal{V}^2(e, k)$ ways of spelling two paths from a node at level k to the root (each path corresponding to one part of a structured model) allowing for up to e substitutions in each.

The total number of operations needed to modify the first k levels of the suffix tree $\mathcal{GT}$ to obtain $\mathcal{GT}'$ before the identification of a second part at a right distance of the first is upper-bounded by

$$\sum_{i=d_{min}}^{d_{max}} N n_{2k+i} \mathcal{V}(e, k) + N n_k \mathcal{V}(e, k) \leq \min\{2, \Delta\} N n_{2k+d_{max}} \mathcal{V}(e, k),$$

where the first summand corresponds to the visits to nodes z coming from w for all m_1 and the second one corresponds to the propagations from z to the root for all m_1.

Restoring $\mathcal{GT}$ to start the extraction of another structured model from a different first part takes $O(N n_k \mathcal{V}(e, k))$ operations using $O(N n_k)$ additional space (size of array L, each cell possibly pointing to a node at level k in $\mathcal{GT}$ or to nil). The total time complexity of the second algorithm is therefore $O(N n_k \mathcal{V}^2(e, k) + \min\{2, \Delta\} N n_{2k+d_{max}} \mathcal{V}(e, k) + N n_k \mathcal{V}(e, k))$. This results in an $O(N n_k \mathcal{V}^2(e, k) + N n_{2k+d_{max}} \mathcal{V}(e, k))$ time complexity. Space complexity is slightly higher than for the first algorithm: $O(N^2 n + N n_k)$ where $n_k \leq Nn$. The second term is for array L.

In either case, the complexity obtained is better both in terms of time and space than the one given by a naive approach to Problem 1.5.4 (see above).

1.5.4.4 Extending the algorithms to extract structured models with $p > 2$ parts (Problem 1.5.5)

1.5.4.4.1 First algorithm: Jumping in the suffix tree

Extending the first algorithm to extract structured models composed of $p > 2$ parts, that is solving Problem 1.5.5, is immediate. After extracting the first i parts of a structured model $((m_1, \ldots, m_p), (d_{min}, d_{max}))$ for

$1 \leq i < p-1$, one jumps down in the tree $\mathcal{GT}$ (following normal tree arcs) to get to the d_{min}- to d_{max}-descendants of every node-occurrence of $((m_1, \ldots, m_i), (d_{min}, d_{max}))$ then continues the extraction from there using Lemma 1.5.5.

A pseudo-code is given below.

```
procedure ExtractModels(Model m, Block i)
1.   for each node-occurrence v of m do
2.       if i > 1 then
3.           put in PotentialStarts the children w of v at levels
             (i − 1)k + (i − 1)d_min to (i − 1)k + (i − 1)d_max
4.       else
5.           put v (the root) in PotentialStarts
6.   for each model m_i (and its occurrences) obtained by doing a recursive
             depth-first traversal from the root of the virtual model tree M
             while simultaneously traversing GT from the node-occurrences
             in PotentialStarts (Lemma 1.5.5 and quorum constraint) do
7.       if i < p then
8.           ExtractModels(m = m_1 ··· m_i, i + 1)
9.       else
10.          report the complete model m = ((m_1, ··· , m_p), (d_min, d_max))
             as valid
```

1.5.4.4.2 Second Algorithm: Modifying the Suffix Tree

Extending the second algorithm to solve Problem 1.5.5 is slightly more complex and thus calls for a few remarks. The operations done to modify the tree between building $m_{i\geq 1}$ and m_{i+1} are almost the same as those described in Section 1.5.4.2.3 except for two facts. One is that up to $(p-1)$ arrays L are now needed to restore the tree after each modification it undergoes. The second, more important difference is that we need to keep for each node v_k at level k reached from an ascent up $\mathcal{GT}$'s suffix links a list, noted $Lptr_{v_k}$, of pointers to the nodes at lower levels that affected the contents of v_k. The reason for this is that tree $\mathcal{GT}$ is modified up to level k only (resulting in tree $\mathcal{GT}'$) as these are the only levels concerned by the search for occurrences of each box of a structured model. Lower levels of $\mathcal{GT}$ remain unchanged, in particular the boolean arrays at each node below level k. To obtain the correct information concerning the potential end node-occurrences of boxes i for $i > 2$ (*i.e.* to which strings such occurrences belong), we therefore cannot descend $\mathcal{GT}$ from the ends of node-occurrences in $\mathcal{GT}'$ of box $(i-1)$. If we did, we would not miss any occurrence but we could get more, *e.g.* ones that did not have an occurrence of a previous box in the model. We might thus overcount some strings and consider as valid a model which, in fact, no longer satisfied the quorum. We have to go down $\mathcal{GT}$ from the ends of node-occurrences *in* $\mathcal{GT}$, that is from the original ends of node-occurrences in $\mathcal{GT}$ of the boxes built so far. These are reached from the list of pointers $Lptr_{v_k}$ for the nodes v_k that are identified as occurrences of the box currently just treated. For models composed of p boxes, we need at most $(p-1)$ lists $Lptr_{v_k}$ for each node v_k at level k.

A pseudo-code for the algorithm is as follows:

procedure ExtractModels(Model m, Block i)
for each node-occurrence v of m **do**
    if $i > 2$ **then**
        put in $PotentialEnds$ the children w at levels $ik + (i-1)d_{min}$ to $ik + (i-1)d_{max}$
        for each node-occurrence w in $PotentialEnds$ **do**
            follow fast suffix-link to node z at level k
            put z in $L(i)$
            if first time z is reached **then**
                initialize $colors_z$ with zero
                put z in $NextEnds$
            add $colors_w$ to $colors_z$
        do a depth-first traversal of $\mathcal{GT}$ to update the boolean arrays from the root to all z in $NextEnds$ (let $\mathcal{GT}'$ be the k-deep tree obtained by such an operation)
    if $i = 1$ **then**
        Tree $= \mathcal{GT}$
    else
        Tree $= \mathcal{GT}'$
for each model m_i (and its occurrences) obtained by doing a recursive depth-first traversal from the root of the virtual model tree $\mathcal{M}$ while simultaneously traversing *Tree* from the root (Lemma 1.5.5 and quorum constraint) **do**
    if $i < p$ **then**
        ExtractModels($m = m_1 \cdots m_i$, $i+1$)
    else
        report the complete model $m = ((m_1, \cdots, m_p), (d_{min}, d_{max}))$ as valid
    if $i > 1$ **then**
        restore tree $\mathcal{GT}$ to its original state using $L(i)$

1.5.4.4.3 Complexity

Using a same reasoning as before, it is not difficult to see that the first algorithm requires $O(Nn_{pk+(p-1)d_{max}}\ \mathcal{V}^p(e,k))$ time, where $\mathcal{V}^p(e,k)) \leq k^{pe}|\Sigma|^{pe}$. The space complexity remains the same as for solving Problem 1, that is $O(N^2 n)$.

In the case of the second algorithm, the p single models composing a structured model may be built in a number of operations upper bounded by $O(Nn_k\ \mathcal{V}^p(e,k))$.

The total number of operations needed to modify the first k levels of the suffix tree $\mathcal{GT}$ to obtain $\mathcal{GT}'$ before the identification of a box $(i+1)$ for $i > 2$ at a right distance of box i is upper-bounded by:

$$\textstyle\sum_{j=d_{min}}^{d_{max}} Nn_{ik+(i-1)j}\mathcal{V}^{i-1}(e,k)) + (Nn_k\mathcal{V}(e,k),$$

which is at most

$$\min\{2,\Delta\}Nn_{ik+(i-1)d_{max}}\mathcal{V}^{i-1}(e,k).$$

Restoring $\mathcal{GT}'$ as we back off to the preceding box takes, as before, $O(Nn_k\ \mathcal{V}(e,k))$ operations using $O(N(p-1)n_k)$ additional space (size of arrays $L(1)$ to $L(p)$).

The total time complexity of Algorithm 2 is therefore of $O(Nn_k\mathcal{V}^p(e,k) + Nn_{pk+(p-1)d_{max}}\ \mathcal{V}^{p-1}(e,k))$. The space complexity is $O(N^2n + N(p-1)n_k)$.

1.5.4.5 Extending the algorithms to handle restricted intervals of unknown limits (Problem 1.5.6)

1.5.4.5.1 First algorithm

In the case where the distances between the two parts m_1 and m_2 of a single model vary inside a restricted interval whose limits are unknown, the first algorithm is extended in the following way. Once a first part m_1 of a structured model $((m_1, m_2), (d_{min}, d_{max}, \delta))$ has been extracted, we jump as before to nodes w in V_2 given in Section 1.5.4.2.2. To verify, as we now must, that:

- there exists d, with $d_{min} + \delta \leq d \leq d_{max} - \delta$, such that level($w$) - level($v$) is equal to $d \pm \delta$;
- (more particularly) d is *the same* for pairs of occurrences (one occurrence for each part of the structured model) present in at least q *distinct strings*;

we just need to keep at each node its distance from level k and to count the number of distinct strings for each restricted interval $d \pm \delta$ separately.

1.5.4.5.2 Second algorithm

In order to verify the same two points mentioned above in the case of the second algorithm, we need this time to keep an additional information at the nodes z situated at level k that are reached from w by jumping back up the tree (following suffix links). This information is required because a node at level k may be reached from nodes w corresponding to different distances from occurrences of the previous box. We therefore need to have at each node z an array of dimension not N but $((d_{max} - d_{min} - (2*\delta)) \times N)$.

The node-occurrences at each extension step of the second part of a model are added for each cell $i \in (d_{max} - d_{min} - (2 * \delta))$ in turn. If for any i, this number is at least q, the model is valid and the second part may be further extended (if its length is still smaller than k).

We denote this boolean array $Colors_z$ with a capital C to stress that it is now multi-dimensional. If it is the first time a node z is reached from w, the l cells of $Colors_z$ for $l \in [\max\{d_{min}+\delta, \text{level}(w)-\text{level}(z)-\delta\}, \min\{d_{max}-\delta, \text{level}(w)-\text{level}(z)+\delta\}]$ are set equal to $colors_w$ and all the other cells are initialized to zero, otherwise $colors_w$ is added (boolean "or") to the l cells of $Colors_z$. Once all nodes v and w have been treated, the information contained in the nodes z that were reached during this operation are propagated up the tree from level k to the root (using normal tree arcs) in the following way: if $\bar{z}$ and $\hat{z}$ have same parent z, then, for all l such that $d_{min} + \delta_1 \leq l \leq d_{max} - \delta_1$, $Colors_z[l] = Colors_{\bar{z}}[l] \cup Colors_{\hat{z}}[l]$.

1.5.4.5.3 Complexity

The time complexity of the first algorithm for solving Problem 1.5.6 remains $O(Nn_{2k+d_{max}}\ \mathcal{V}^2(e,k))$ and the space complexity $O(nN^2)$.

The time complexity of the second algorithm for solving the same problem becomes $O(N\ \Delta'\ n_k\mathcal{V}^2(e,k) + N\Delta' n_{2k+d_{max}}\ \mathcal{V}(e,k))$ where $\Delta' = d_{max} - d_{min} - (2 * \delta)$. The space complexity is $O(N^2 n + N\Delta' n_k)$.

1.5.4.6 Extending the algorithms to extract structured models with $p > 2$ parts and to handle restricted intervals of unknown limits (Problem 1.5.7)

Few changes are required when one wishes to consider structured models that are composed of more than two boxes separated by intervals of distances of the type $d \pm \delta$ for some d and a fixed δ. The main one concerns the second algorithm: the boolean arrays at each node in the suffix tree have now to be of dimension $N(p-1)\Delta'$. The Δ' comes from having to handle restricted intervals of unknown limits as we saw in Section 1.5.4.5. The $(p-1)$ comes from the fact that d may be different for each pair of successive boxes in the structured model. The time and space complexity will therefore be further multiplied by a term of $(p-1)$.

The time complexity of the second algorithm (the only one for which there is a change) for solving Problem 1.5.7 is $O(N\Delta'(p-1)n_k\mathcal{V}^p(e,k) + N\Delta'(p-1)n_{pk+(p-1)d_{max}}\ \mathcal{V}^{p-1}(e,k))$. The space complexity is $O(N^2 n + N\Delta'(p-1)n_k)$.

1.5.5 Satellites

1.5.5.1 The satellite problem

The satellite problem we propose to solve is the following:

Problem 1.5.8 *Given a string s and parameters min_repeat, min_range, max_range, max_jump, and e (possibly also g), find all consensus satellite models m that are valid for s.*

In fact, the original papers [34, 35] reported a set of disjoint "fittest" trains realizing each model m given a measure of "fitness".

The algorithm presented below is the only combinatorial, non-heuristical developed so far for identifying tandem arrays. Other exact approaches either treated the case of tandem repeats only [22, 19], did not allow for errors [3, 5, 26, 46] or required generating all possible (not just valid) models of a given length [8, 10, 32].

1.5.5.2 Building prefix satellite models

1.5.5.2.1 Main ideas

As with all previous cases considered in this paper, satellite models are constructed by increasing lengths. In order to determine if a model is valid, we must have some representation of the train or wagons that make it so. There are two possibilities:

- we can keep track of each validating train and its associated wagons, or
- we can keep track of individual wagons, and, on the fly, determine if they can be combined into validating trains.

The first possibility is appealing because model extension is straightforward. We would just have to check, for each wagon of each train, whether it can be extended in relation to the extended model, and then count how many wagons remain to see whether the train it belonged to is still a valid one. However, there are generally many overlapping trains involving many of the same wagons for a given model. Common wagons may be present more than once in the list of occurrences of m if this is kept as a list of trains. This approach entails redundancies that lead to an inefficient algorithm. We therefore adopt the second approach, of keeping track of wagons and determining if they can be assembled into trains as needed.

The rules of prefix-model extension are given in Lemma 1.5.6 below. A wagon is identified by a triple (i, j, d) indicating that it is the substring $s_i s_{i+1} \ldots s_j$ of s and is $d \leq e$ differences away from its model. Position i indicates the left-end of the wagon, and j its right-end. Contrary to the previous algorithms presented in this paper, models and their occurrences (the wagons) will be extended to the left. This is just to facilitate verifying Property 1.4.3. Right ends of occurrences are calculated but will be used only for checking Property 1.4.4.

Lemma 1.5.6 *The triple (i, j, d) encodes a wagon of $m' = \alpha m$ with $\alpha \in \Sigma$ and $m \in \Sigma^k$ if, and only if, $d \leq e$ and at least one of the following conditions is true:*

(match) $(i+1, j, d)$ *is a wagon of* m *and* $s_i = \alpha$;
(substitution) $(i+1, j, d-1)$ *is a wagon of* m *and* $s_i \neq \alpha$;
(deletion) $(i, j, d-1)$ *is a wagon of* m;
(insertion) $(i+1, j, d-1)$ *is a wagon of* αm.

For each prefix-model m, we keep a list of the wagons of m that are in at least one train validating m. We describe such wagons as being valid with respect to m. When we extend a model (to the left) to $m' = \alpha m$, we perform two tasks:

- First, determine which valid wagons of m can be extended as above to become wagons of m'.
- Second, of these newly determined wagons of m', keep only those that are valid with respect to m'. This requires effectively assemblying wagons into trains, something that is not needed in an approach that would keep track of trains directly.

Note that we need not actually enumerate the trains in the second step, we simply must determine if a wagon is part of one. This will allow us to perform an extension step in time linear with the length of the string.

As a final insight, consider the directed graph $G = (V, E)$ where V is the set of all valid wagons and there is an edge from wagon u to v if $left_v - left_u \in JUMP$. Then a wagon u is valid if it is part of a path of length min_repeat or more in G. Determining this is quite simple as the graph is clearly acyclic. In the computation that will follow, we will effectively compute the length of the longest path to u in $Lcnt_u$ and the length of the longest path from u in $Rcnt_u$. If $Lcnt_u + Rcnt_u > min\_repeat$ then u is valid.

1.5.5.3 Consensus satellite models

1.5.5.3.1 Main ideas

We encode the collection of all wagons of m in a set, $L_m \subseteq \{1 \ldots, n\}$, and an $(n+1) \times (2e+1)$-element array D_m as follows:

1. $i \in L_m$ if and only if i is the left-end of at least one wagon valid with respect to m.
2. for each $i \in L_m$, the value $D_m[i, \delta]$ for $\delta \in [-e, e]$ is the edit distance of m from wagon $s_i s_{i+1} \ldots s_{i+|m|-1+\delta}$.

Intuitively, L_m gives the left-ends of all valid wagons which is all we need to check Properties 1.4.2 and 1.4.3. D_m gives us the distances we need for extending models, together with the right-ends needed for checking Property 1.4.4. Formally, $(i, i+|m|-1+\delta, d)$ is a valid wagon of m if and only if $i \in L_m$ and $d = D_m[i, \delta] \leq e$.

The complete algorithm is given below. When Extend(αm) is called, it is assumed that L_m is known along with the relevant D_m values. The routine computes these items for the extension αm and recursively for the extensions thereof. Lines 0-5 compute the set of left-ends of wagons for αm derivable from wagons of m that are valid. While Lemma 1.5.6 gives us a way to do so, recall that we are using dynamic programming to compute all extensions simultaneously. This corresponds to adding the last row to the dynamic programming matrix of s versus αm. At the start L_m gives all the positions in row $|m|$ that have value e or less (and are valid) and D_m gives their values. From these, we compute the positions in row $|m|+1$ in the obvious sparse fashion to arrive at $L_{\alpha m}$ and the values $D_{\alpha m}$.

int $Lcnt[1..n]$, $Rcnt[1..n]$

procedure Extend(αm)

    $L_{\alpha m} \leftarrow \emptyset$
1. **for** $i+1 \in L_m$ (in decreasing order) **do**
2.   **for** $\delta \in [-e, e]$ **do**
3. $D_{\alpha m}[i,\delta] \leftarrow \min \left\{ \begin{array}{l} D_m[i+1,\delta] + (\textbf{if } s_i = \alpha \textbf{ then } 0 \textbf{ else } 1), \\ \textbf{if } i \in L_m \textbf{ then } D_m[i,\delta+1]+1, \\ \textbf{if } i{+}1 \in L_{\alpha m} \textbf{ then } D_{\alpha m}[i+1,\delta-1]+1 \end{array} \right\}$
4.   **if** $min_\delta\{D_{\alpha m}[i,\delta]\} \leq e$ **then**
5. $L_{\alpha m} \leftarrow L_{\alpha m} \cup \{i\}$
6. **for** $i \in L_{\alpha m}$ (in decreasing order) **do**
7. $Rcnt[i] \leftarrow \max_{k \in (i+JUMP) \cap L_{\alpha m}} \{Rcnt[k]\} + 1$
8. **for** $i \in L_{\alpha m}$ (in increasing order) **do**
9. $Lcnt[i] \leftarrow \max_{k \in (i-JUMP) \cap L_{\alpha m}} \{Lcnt[k]\} + 1$
10. **for** $i \in L_{\alpha m}$ **do**
11.   **if** $Lcnt[i] + Rcnt[i] \leq min\_repeat$ **then** $L_{\alpha m} \leftarrow L_{\alpha m} - \{i\}$
12. **if** $L_{\alpha m} \neq \emptyset$ **then**
13.   **if** $|\alpha m| \in [min\_range, max\_range]$ **then**
14.   Record(αm)
15.   **if** $|\alpha m| < max\_range$ **then**
16.   **for** $\beta \in \Sigma$ **do**
17.   Extend($\beta\alpha m$)

Once wagons have been extended when possible, we have to eliminate those that are no longer valid. This is performed by Lines 6 to 11. We compute, for each position $i \in L_{\alpha m}$, the maximum number of wagons in a train starting with a wagon whose left-end is at i in $Rcnt[i]$ (including itself), and the maximum number of wagons in a train ending with a wagon whose left-end is at i in $Lcnt[i]$. The necessary recurrences are given in Lines 7 and 9 of the algorithm where we recall that $JUMP = \{y : y \in \bigcup_{x \in [1, max\_jump]} x \times [min\_range, max\_range]\}$ and $i+JUMP$ denotes adding

i to each element of *JUMP*. Observe that $Rcnt[i] + Lcnt[i] - 1$ is the length of the longest train containing a wagon whose left-end is at position i.

Clearly Lines 6-9 take $O(|L_{\alpha m}||JUMP|)$ time. However, when $L_{\alpha m}$ is a very large fraction of n, one can maintain an $Rcnt(Lcnt)$-prioritized queue of the positions in $(i + JUMP) \cap L_{\alpha m}$, to obtain an $O(n \cdot max\_jump \cdot \log |JUMP|)$ bound.

Finally in the remaining steps, Lines 12-17, the algorithm calls Record to record potential models and then recursively tries to extend the model if possible. The routine Record confirms that the model is a consensus model by checking Property 4.3 and recording the intervals spanned by trains valid for the consensus model, if any.

1.5.5.4 Complexity

We start by observing that $O(|JUMP| + e)$ time is spent on a given left-end position for each prefix model matching the string beginning at that position. The term e comes from Lines 2 and 3 while $|JUMP|$ is the number of back or forward elements that have to be examined in order to determine, for each wagon, the length of the longest train it may belong to (Lines 9 and 11). The number of prefix models that could match a given position with e or fewer errors is by definition $\Sigma_{k=1}^{max\_range} \mathcal{V}(e, k)$. The total time taken by the algorithm is therefore bounded above by $O(n \cdot (|JUMP| + e) \cdot max\_range \cdot \mathcal{V}(e, max\_range)) = O(n \cdot max\_range^2 \cdot max\_jump \cdot \mathcal{V}(e, max\_range))$ as $e < max\_range$.

The space requirement is that of keeping all the information concerning at most max_range models at a time (a model m and all its prefixes). It is therefore on the order of at most $O(n \cdot max\_range \cdot e)$ as only $O(ne)$ storage is required to record the left-end positions and edit-distance at each possible right-end.

1.5.5.5 By-Products and Extension

1.5.5.5.1 By-Products

One of the by-products of the approach for finding satellites described above is that the algorithm is also capable of identifying periodic, or approximately periodic repeats that may be non-contiguous. These correspond to prefix models, *i.e.* to models that verify properties 1.4.2 and 1.4.3 but not necessarily Property 1.4.4.

1.5.5.5.2 Tandem Direct and Inverted Repeats

A more substantial modification of the algorithm permits us to treat the presence of inverted repeats amongst the direct ones when these occur in tandem. The exercise of doing so entails distinguishing between direct and inverted wagons, extending properties 1.4.2 and 1.4.3 to accommodate the inverted wagons, and then modifying the basic algorithm of Section 1.5.5.3.1 accordingly.

The extension of properties 1.4.3 and 1.4.4 gives raise to the following new versions:

Property (1.4.3)'

$$end_{u_{i+1}} - end_{u_i} \in \begin{cases} JUMP, & \text{if both } u_i \text{ and } u_{i+1} \text{ are direct} \\ & \text{or both are inverted;} \\ JUMP\_INV, & \text{otherwise;} \end{cases}$$

where

$$end_{u_{i+1}} - end_{u_i} \in \begin{cases} left_u, & \text{if } u \text{ is direct;} \\ right_u, & \text{otherwise;} \end{cases}$$

and

$$JUMP\_INV = \bigcup_{x \in [0, max\_jump-1]} [x \times min\_range, (x+2) \times max\_range].$$

Property (1.4.4)'

$$start_{u_{i+1}} - end_{u_i} \in GAP,$$

where

$$start_u = \begin{cases} right_u, & \text{if } u \text{ is direct;} \\ left_u, & \text{otherwise;} \end{cases}$$

and

$$end_u = \begin{cases} left_u, & \text{if } u \text{ is direct;} \\ right_u, & \text{otherwise.} \end{cases}$$

A wagon is now identified by a quadruple (i, j, d, f) where the additional variable f is a flag indicating whether we are dealing with a direct or an inverted occurrence. Further details may be found in [34].

References

[1] P. Bieganski, J. Riedl, J. V. Carlis, and E.M. Retzel.Generalized suffix trees for biological sequence data: applications and implementations.In *Proc. of the 27th Hawai Int. Conf. on Systems Sci.*, pages 35–44. IEEE Computer Society Press, 1994.

[2] B. Charlesworth, P. Sniegowski, and W. Stephan.The evolutionary dynamics of repetitive DNA in eukaryotes.*Nature*, 371:215–220, 1994.

[3] B. Clift, D. Haussler, R. McConnell, T. D. Schneider, and G. D. Stormo.Sequence landscapes.*Nucleic Acids Res.*, 14:141–158, 1986.

[4] T. E. Creighton.In *Proteins, Structures and Molecular Properties*. Freeman, 1993.

[5] M. Crochemore.An optimal algorithm for computing the repetitions in a word.*Inf. Proc. Letters*, 12:244–250, 1981.

[6] M. Crochemore and W. Rytter.*Text algorithms*.Oxford University Press, 1994.

[7] M.O. Dayhoff, R.M. Schwartz, and B.C. Orcutt.A model of evolutionary change in proteins.In M.O. Dayhoff, editor, *Atlas of Protein Sequence an Structure*, volume 5 suppl.3, pages 345–352. Natl. Biomed. Res. Found., 1978.

[8] O. Delgrange.*Un algorithme rapide pour une compression modulaire optimale. Application à l'analyse de séquences génétiques*.PhD thesis, 1997.Thèse de doctorat - Université de Lille I.

[9] V. Escalier, J. Pothier, H. Soldano, and A. Viari.Pairwise and multiple identification of three dimensional common substructures in proteins.*J. Computational Biology*, 1996.

[10] V. Fischetti, G. Landau, J. Schmidt, and P. Sellers.Identifying periodic occurrences of a template with applications to protein structure.In Z. Galil A. Apostolico, M. Crochemore and U. Manber, editors, *Combinatorial Pattern Matching*, volume 644 of *Lecture Notes in Computer Science*, pages 111–120. Springer-Verlag, 1992.

[11] Y. M. Fraenkel, Y. Mandel, D. Friedberg, and H. Margalit.Identification of common motifs in unaligned DNA sequences: application to *escherichia coli lrp* regulon.*Comput. Appl. Biosci.*, 11:379–387, 1995.

[12] D. J. Galas, M. Eggert, and M. S. Waterman.Rigorous pattern-recognition methods for DNA sequences. analysis of promoter sequences from escherichia coli.*J. Mol. Biol.*, 186:117–128, 1985.

[13] D. Gusfield.*Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*.Cambridge University Press, 1997.

[14] S. Henikoff and J.G. Henikoff.Amino acid substitution matrices from protein blocks.*Proc. Natl. Acad. Sci. USA*, 89:10915–10919, 1992.

[15] L. C. K. Hui.Color set size problem with applications to string matching.In A. Apostolico, M. Crochemore, Z. Galil, and U. Manber, editors, *Combinatorial Pattern Matching*, volume 644 of *Lecture Notes in Computer Science*, pages 230–243. Springer-Verlag, 1992.

[16] I. Jonassen.Efficient discovery of conserved patterns using a pattern graph.*Comput. Appl. Biosci.*, 13:509–522, 1997.

[17] I. Jonassen, J. F. Collins, and D. G. Higgins.Finding flexible patterns in unaligned protein sequences.*Protein Science*, 4:1587–1595, 1995.

[18] I. Jonassen, I. Eidhammer, and W. R. Taylor.Discovery of local packing motifs in protein structures.*Proteins: Structure, Function, and Genetics*, 34:206–219, 1999.

[19] S. K. Kannan and E. W. Myers.An algorithm for locating non-overlapping regions of maximum alignment score.In Z. Galil A. Apostolico, M. Crochemore and U. Manber, editors, *Combinatorial Pattern Matching*, volume 684 of *Lecture Notes in Computer Science*, page 7486. Springer-Verlag, 1993.

[20] R.M. Karp, R.E. Miller, and A.L. Rosenberg.Rapid identification of repeated patterns in strings, trees and arrays.pages 125–136. Proc. 4th Annu. ACM Symp. Theory of Computing, 1972.

[21] A. Klingenhoff, K. Frech, K. Quandt, and T. Werner.Functional promoter modules can be detected by formal models independent of overall nucleotide sequence similarity.*Bioinformatics 1*, 15:180–186, 1999.

[22] G. Landau and J. Schmidt.An algorithm for approximate tandem repeats.In Z. Galil A. Apostolico, M. Crochemore and U. Manber, editors, *Combinatorial Pattern Matching*, volume 684 of *Lecture Notes in Computer Science*, pages 120–133. Springer-Verlag, 1993.

[23] L. Marsan and M.-F. Sagot.Algorithms for extracting structured motifs using a suffix tree with an application to promoter and regulatory site consensus identification.*J. Computational Biology*, 7:345–362, 2000.

[24] L. Marsan and M.-F. Sagot.Extracting structured motifs using a suffix tree – algorithms and application to promoter consensus identification.In S. Istrail, P. Pevzner, and M. Waterman, editors, *RECOMB'00. Proceedings of Fourth Annual International Conference on Computational Molecular Biology.* ACM Press, 2000.

[25] E. M. McCreight.A space-economical suffix tree construction algorithm.*J. ACM*, 23:262–272, 1976.

[26] A. Milosavljevic and J. Jurka.Discovering simple DNA sequences by the algorithmic significance method.*Comput. Appl. Biosci.*, 9:407–411, 1993.

[27] L. Parida, I. Rigoutsos, A. Floratos, D. Platt, and Y. Gao.Pattern discovery on character sets and real-valued data: linear bound on irredundant motifs and polynomial time algorithms.In *Proc. of the eleventh ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 297–308. ACM Press, 2000.

[28] J. Pothier.1993.Personal communication.

[29] C. Queen, M. N. Wegman, and L. J. Korn.Improvements to a program for DNA analysis: a procedure to find homologies among many sequences.*Nucleic Acids Res.*, 10:449–456, 1982.

[30] G. N. Ramachandran, C. Ramakrishnan, and V. Sasisekharan.Stereochemistry of polypeptide chain configurations.*J. Mol. Biol.*, 87:95–99, 1963.

[31] J.L. Risler, M.O. Delorme, H. Delacroix, and A. Hénaut.Amino acid substitutions in srtucturally related proteins: a pattern recognition approach.*J. Mol. Biol.*, 204:1019–1029, 1988.

[32] E. Rivals and O. Delgrange.A first step toward chromosome analysis by compression algorithms.In N. G. Bourbakis, editor, *First International IEEE Symposium on Intelligence in Neural and Biological Systems*, pages 233–239. IEEE Computer Society Press, 1995.

[33] M.-F. Sagot, V. Escalier, A. Viari, and H. Soldano.Searching for repeated words in a text allowing for mismatches and gaps.In R. Baeza-Yates and U. Manber, editors, *Second South American Workshop on String Processing*, pages 87–100, Viñas del Mar, Chili, 1995. University of Chili.

[34] M.-F. Sagot and E. W. Myers.Identifying satellites and periodic repetitions in biological sequences.*J. of Computational Biology*, 10:10–20, 1998.

[35] M.-F. Sagot and E. W. Myers.Identifying satellites in nucleic acid sequences.In S. Istrail, P. Pevzner, and M. Waterman, editors, *RECOMB'98. Proceedings of Second Annual International Conference on Computational Molecular Biology*, pages 234–242. ACM Press, 1998.

[36] M.-F. Sagot and A. Viari.A double combinatorial approach to discovering patterns in biological sequences.In D. Hirschberg and G. Myers, editors, *Combinatorial Pattern Matching*, volume 1075 of *Lecture Notes in Computer Science*, pages 186–208. Springer-Verlag, 1996.

[37] M. F. Sagot, A. Viari, J. Pothier, and H. Soldano.Finding flexible patterns in a text - an application to 3D molecular matching.*Comput. Appl. Biosci.*, 11:59–70, 1995.

[38] M.-F. Sagot, A. Viari, and H. Soldano.A distance-based block searching algorithm.In C. Rawlings, D. Clark, R. Altman, L. Hunter, T. Lengauer, and S. Wodak, editors, *Third International Symposium on Intelligent Systems for Molecular Biology*, pages 322–331, Cambridge, England, 1995. AAAI Press.

[39] M.-F. Sagot, A. Viari, and H. Soldano.Multiple comparison: a peptide matching approach.*Theoret. Comput. Sci.*, 180:115–137, 1997.presented at *Combinatorial Pattern Matching 1995.*

[40] H. Soldano, A. Viari, and M. Champesme.Searching for flexible repeated patterns using a non transitive similarity relation.*Pattern Recognition Letters*, 16:233–246, 1995.

[41] R. Staden.Methods for discovering novel motifs in nucleic acid sequences.*Comput. Appl. Biosci.*, 5:293–298, 1989.

[42] E. Ukkonen.Constructing suffix trees on-line in linear time.pages 484–492. IFIP'92, 1992.

[43] J. van Helden, A. F. Rios, and J. Collado-Vides.Discovering regulatory elements in non-coding sequences by analysis of spaced dyads.*Nucleic Acids Res.*, 28:1808–1818, 2000.

[44] A. Vanet, L. Marsan, A. Labigne, and M.-F. Sagot.Inferring regulatory elements from a whole genome. An analysis of the σ^{80} family of promoter signals.*J. Mol. Biol.*, 297:335–353, 2000.

[45] A. Vanet, L. Marsan, and M.-F. Sagot.Promoter sequences and algorithmical methods for identifying them.*Research in Microbiology*, 150:779–799, 1999.

[46] R. Verin and M. Crochemore.Direct construction of compact directed acyclic word graphs.In A. Apostolico and J. Hein, editors, *Combinatorial Pattern Matching*, volume 1264 of *Lecture Notes in Computer Science*, pages 116–129. Springer-Verlag, 1997.

[47] M. S. Waterman.General methods of sequence comparison.*Bull. Math. Biol.*, 46:473–500, 1984.

[48] M. S. Waterman.Multiple sequence alignments by consensus.*Nucleic Acids Res.*, 14:9095–9102, 1986.

[49] M. S. Waterman.Consensus patterns in sequences.In M. S. Waterman, editor, *Mathematical Methods for DNA Sequences*, pages 93–116. CRC Press, 1989.