Searching for repeated words in a text allowing for mismatches and gaps

Marie-France Sagot [1,3]
Vincent Escalier [1,4]
Alain Viari [1]
Henri Soldano [1,2]

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

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

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

[4] LITP - URA CNRS 248
Tour 55-56, 4 Place Jussieu 75252 - Paris Cedex 05

Abstract

We present in this paper an algorithm that locates similar words common to a set of strings defined over an alphabet Σ, where the similarity is stated in terms of a Levenshtein edit distance. The comparison of the words in the strings is realized by using a reference object called a model which is a word over Σ. This allows us to perform a multiple comparison of the strings as opposed to pairwise comparisons, and the algorithm is particularly appropriate for the analysis of DNA/RNA sequences.
keywords : multiple comparison, Levenshtein edit distance, model

1 Introduction

A lot has been written on the subject of finding a pattern in a text, whether strictly or flexibly, and efficient algorithms have been elaborated to treat that problem ([1] [2] [3] [5] [10] [11] [12] [14] [13] for some of the references). However, there are situations where we do not have any idea of what must be looked for in a text. Such a situation arises in biology, when we wish to find the similarities existing in a set of molecular sequences, that is, in a set of strings. While in the case of a pattern that is searched for in a text, a pairwise comparison is all we need, when looking for the similarities present in a set of strings, we must be able to perform a multiple comparison of these strings. One way to do that is to look for all the words common to the strings of the set, that is, to look for all the words that are repeated in all the strings or in some of them. In this case too, the search for these common words can be performed in a strict or flexible manner.

Finding all the words common to a set of strings is done by an algorithm called KMR that was elaborated in 1972 [4] and later adapted to this specific purpose [6]. KMR was then extended in 1994 to an algorithm called KMRC [9] that is able to perform this search in a more flexible way. This flexibility is introduced by means of a similarity relation R, in general not transitive, that is established between the symbols of the alphabet over which the strings are defined. Two words of length k are then considered to be similar if the symbols located at the same positions are related by R. The problem of finding approximate repetitions is thus equivalent to that of finding the maximal cliques of the relation induced by R on the words of length k.

KMRC's approach is well suited to the analysis of proteins, where the strings studied represent amino acid sequences, since it is possible to group the amino acids into sets sharing, for instance, the same physico-chemical properties. It also proved to be of interest in the analysis of protein 3D structures [8]. However, the use of a relation of similarity between symbols is meaningless in the case of nucleic acid sequences, for which the notion of approximate rests on the introduction of mismatches and/or gaps. In this case, two words are considered to be similar if the edit distance between them is at most a fixed value e, where the edit distance between two words u and v is that given by Levenshtein : the minimum number of steps (substitutions and insertions/deletions) necessary to convert u to v.

The subject of this paper is therefore to present a different extension of the KMR algorithm, that locates similar words common to a set of strings where the similarity is defined in terms of the Levenshtein edit distance. We do so in two steps, introducing first the Hamming distance (substitutions only), then the full Levenshtein distance. Underlying both versions of the algorithm is an important concept, called a model, we also define here.

Section 2 gives some basic definitions together with an overview of the KMR algorithm. Section 3 introduces the concept of model. Section 4 presents the Hamming version of the new algorithm, called H-Moivre, and discusses its time and space complexities together with one way of improving its performance. Section 5 gives the Levenshtein version, called L-Moivre, and its time and space complexities. Finally, we conclude with general remarks on future work.

2 Basic definitions and earlier algorithms

Let Σ be an alphabet. A string s over Σ is an element of Σ^* and we call $r \in \Sigma^*$ a word in s if $s = u_1 r u_2$ with u_1, $u_2 \in \Sigma^*$ and $u_1 r u_2$ the usual operation of concatenation in Σ^*. We denote $|r|$ the length of r. A repeated word (also called a repetition) in s is a string $r \in \Sigma^+$ such that $s = u_1 r u_2 = v_1 r v_2$ with u_1, u_2, v_1, $v_2 \in \Sigma^*$ and $|u_1| \neq |v_1|$. We wish to find all the words common to a set $\{s1, s2, \ldots , sN\}$ of strings, where the strings represent nucleic acid sequences. This can be done by searching for repeated words in the string $s = s1s2...sN$.

A classical algorithm for finding all exact repetitions in a string (we call it KMR) was elaborated by Karp, Miller and Rosenberg. Given a string s, KMR solves the following problems :

1. identify the positions of all the words of a fixed length k that appear repeated in s;
2. find the length k_{max} of the longest repeated word in s, and solve problem 1 for $k = k_{max}$.

The main idea of KMR rests on the definition of the following equivalence relation over the positions of s :

Definition 2.1 *Given a string $s = s_1 s_2 ... s_n \in \Sigma^n$, two positions i and $j \in \{1,...,n-k+1\}$ of s are said to be k-equivalent, noted $i\ E_k\ j$, if and only if the words of length k starting at these positions in s are identical.*

Given this definition, problems 1 and 2 can be formulated again as the problem of finding the partition of E_k. Each class of E_k of cardinality greater than or equal to two represents a set of exactly repeated words in s. The algorithm is based on the iterative construction of these partitions, starting with E_1, then using E_1 to build E_2, etc. The mechanism for performing such constructions rests on the following lemmas :

Lemma 2.1 *Let $s = s_1 s_2 ... s_n \in \Sigma^n$. Let a, $b \in \{1,...,n\}$ with $b \leq a$, and i, $j \in \{1,...,n-(a+b)+1\}$. Then:*

$$i\ E_{a+b}\ j \quad \text{if and only if} \quad i\ E_a\ j \quad \text{and} \quad i+b\ E_a\ j+b.$$

Lemma 2.2 *Let $s = s_1 s_2 ... s_n \in \Sigma^n$. Let a, $b \in \{1,...,n\}$ and i, $j \in \{1,...,n-(a+b)+1\}$. Then :*

$$i\ E_{a+b}\ j \quad \text{if and only if} \quad i\ E_a\ j \quad \text{and} \quad i+a\ E_b\ j+a.$$

In practice, either lemma is used with $a = b$ to iteratively compute the exactly repeated words of length 2, 4, 8, etc. If k (or k_{max}) is not a power of 2, we first find the words whose length $k' < k$ is a power of 2, then use lemma 2.2 to put the parts together.

What these lemmas show is that the construction of the relation E_k and of its classes is basically an operation of set intersection on the classes of $E_{k'}$ for $k' < k$.

These intersections are implemented using stacks of integers. More precisely, as shown in figure 1, we need an array of size n which stores, for each position i in s, the label of the class of E_k to which the k-length word starting at i belongs. Then the application of a lemma is done by means of two arrays of stacks P and Q which contain for each class c the positions i belonging to c. Details of the algorithm can be found in the original paper [4]. KMR time complexity is $O(n.\log(k))$, its space complexity is $O(n)$.

The two versions of the new algorithm presented in sections 4 and 5 are variants of KMR. They also realize operations of intersections on sets, each element of which identifies a word in s, and characterizes the relation existing between this word and an object called a model.

3 Models

In KMR, the search for exactly repeated words in a string s was done by comparing these words between themselves. To find all approximately repeated words in a string s we proceed in a different manner. This time we compare words in an indirect way by referencing them to an object called a model we presently define.

Definition 3.1 *A model m of length k is an element of Σ^k.*

Notation 3.1 *Given $u \in \Sigma^k$ and a model m of length k, we call edit distance between u and m the minimum number of :*

1. *substitutions only (Hamming distance);*

2. *substitutions and insertions/deletions (Levenshtein distance)*

that is necessary to convert u to m.

We note these distances H-distance(u,m) and L-distance(u,m) respectively.

Definition 3.2 *Given a model m, m is said to be H-present in s at location i if the word u of length $\mid m \mid$ starting at position i in s is such that H-distance(u,m) $\leq e$. We call u an H-image of m.*

Let d_s = H-distance(u,m) be the number of substitutions between u and m.

We say that (i,d_s) is an H-occurrence of m in s.

We note $H(m)$ the set of all H-occurrences of m in s, that is :

$$H(m) = \{(i,d_s) \mid d_s \leq e\}.$$

Example 3.1 *Let m = abab, s = ababbba and e = 1. Then $H(m) = \{(1,0), (3,1)\}$.*

Definition 3.3 *Given a set $H(m)$ of H-occurrences in s containing at least two elements, we call M-Hamming repetitions the words of length $\mid m \mid$ starting at position i in s for all $(i,d_s) \in H(m)$.*

Definition 3.4 *Given a model m, m is said to be L-present in s at location i if there exists at least one word u starting at position i in s such that L-distance(u,m) $\leq e$. We call u an L-image of m.*

Let $d_s + d_d + d_i$ = L-distance(u,m) where d_s, d_d and d_i are the number of substitutions, deletions and insertions respectively.

We say that (i,d_s,d_d,d_i) is an L-occurrence of m in s. We have $\mid u \mid = \mid m \mid - d_d + d_i$.

Finally, we note $L(m)$ the set of all L-occurrences of m in s, that is :

$$L(m) = \{(i,d_s,d_d,d_i) \mid d = d_s+d_d+d_i \text{ is a Levenshtein distance and } d \leq e\}.$$

Example 3.2 *Let m = abab, s = ababbba and e = 1. Then $L(m) = \{(1,0,0,0), (1,0,1,0), (1,0,0,1), (2,0,1,0), (3,1,0,0), (3,0,1,0)\}$.*

Notice that a model may be L-present in s at location i more than once, with different L-images. It could even be present for the same image u with different L-occurrences.

Definition 3.5 *Given a set $L(m)$ of L-occurrences in s containing at least two elements with different positions i, we call M-Levenshtein repetitions the words of length $\mid m \mid -d_d+d_i$ starting at position i in s for all $(i,d_s,d_d,d_i) \in L(m)$.*

In both cases, the relation that is established between a model and its images induces a relation between the images themselves, that is between the words of s, since the edit distance between any two of them is at most $2e$. However, given $p > 2$ words of s, u_1,u_2,...,u_p, such that H/L-distance(u_i,u_j) $\leq 2e$ for all $i, j \in \{1,2,...,p\}$, there may not exist any model of which u_1,u_2,...,u_p are all H/L-images.

Finally, let us observe that given $p \geq 2$ words of s, u_1,u_2,...,u_p, such that this time H/L-distance(u_i,u_j) $\leq e$ for all $i, j \in \{1,2,...,p\}$, then any u_i is a model of which u_1,u_2,...,u_p are all H/L-images.

4 M-Hamming repetitions

4.1 Presentation of H-Moivre

The H-Moivre algorithm is defined in terms of the Hamming distance. Given parameters e and q and a string s, H-Moivre is able to solve the following problems :

1. find all models, and their occurrences, of a given length k that are H-present at least q times in s;
2. find the length k_{max} for which there exists at least one model of length k_{max} that is H-present at least q times in s, and solve problem 1 above for $k = k_{max}$.

Problems 1 and 2 are equivalent to finding all the sets $H(m)$ containing at least q elements, that is, all M-Hamming repetitions of length k, where k is either a fixed value or is equal to k_{max}.

To solve these problems, H-Moivre uses the following lemmas. Let s be a string of length n. Let m, m_1, m_2 be three models such that $m = m_1m_2$ and $i \in \{1,...,n\text{-}|m|\text{+}1\}$ a position in s. Then :

Lemma 4.1

$$(i, d_s) \in H(m)$$
$$\Longrightarrow$$
$$\exists\,!\; d_{s_1}, d_{s_2} \textit{ such that}$$
$$(i, d_{s_1}) \in H(m_1),\ (i+ \mid m_1 \mid, d_{s_2}) \in H(m_2)$$
$$\textit{and } d_s = d_{s_1} + d_{s_2}.$$

Lemma 4.2

$$(i, d_{s_1}) \in H(m_1),\ (i+ \mid m_1 \mid, d_{s_2}) \in H(m_2)$$
$$\textit{such that } \ d_{s_1} + d_{s_2} \leq e$$
$$\Longrightarrow$$
$$(i, d_{s_1} + d_{s_2}) \in H(m_1m_2 = m).$$

These lemmas are the equivalent of lemma 2.2 of KMR. Note that the equivalent of lemma 2.1 does not hold in this case.

These lemmas give a constructive property to build $H(m)$ from $H(m_1)$ and $H(m_2)$. Before we give this property, let us introduce a notation and a special operation of intersection on the sets of H-occurrences of two models m_1 and m_2.

Notation 4.1 *Given $H(m)$ a set of H-occurrences, we note :*

$$(H(m))_{-b} = \{(i{,}d_s) \mid (i + b{,}d_s) \in H(m)\}.$$

This is simply the set obtained from $H(m)$ by subtracting b to each i in $H(m)$.

Definition 4.1 *Given two sets of H-occurrences $H(m_1)$ and $H(m_2)$, we define the operation $\sqcap_H$ by :*

$$H(m_1) \sqcap_H H(m_2) = \{(i,d_s) \mid (i,d_{s_1}) \in H(m_1),\ (i,d_{s_2}) \in H(m_2)$$
$$\textit{and } d_s = d_{s_1} + d_{s_2} \leq e\}.$$

Then from the lemmas it comes :

Proposition 4.1 $H(m) = H(m_1) \sqcap_H (H(m_2))_{-|m_1|}$.

Using this property, the algorithm proceeds in an iterative way to construct first the models of length 1, then uses these models of length 1 to construct those of length 2, etc. The length of the models constructed can be doubled at each step, or simply incremented by one. We shall see which method is best when we discuss the space complexity of the algorithm.

As in KMR, these constructions are basically operations of set intersections, the sets being now composed of occurrences (i,d_s) of models. The main difference with KMR is that a position i can occur in more than one model (which is not case with classes in KMR).

The simple array V_k of KMR thus becomes an array of stacks that contains, at position i, the labels and H-distances to all models m which occur at i (see figure 2). The two arrays of stacks P and Q play the same role as in KMR, except that they now contain pairs (i, d_s). The application of property 4.1 is performed in a similar way as in KMR. The occurrences whose H-distance is greater than e are removed at this step.

Finally, let us recall that our purpose is not to find <u>all</u> models but only those which are repeated at least q times (or on q different substrings in the case of concatenated strings). As in KMR, this is performed by removing during the construction the models which do not occur at least q times. This strategy is justified since no longer model based on them can occur more than q times.

Observe that, at the end of each step, it may occur that several models share exactly the same positions in s. Their occurrences could differ either by the number or the positions of the mismatches, or could even be strictly identical. They are all <u>valid</u> models in the sense we defined them. The question to know whether these models should all be kept or if some could be eliminated at the end depends upon each particular application.

4.2 Time and space complexities of H-Moivre

4.2.1 Construction of the models in H-Moivre - a depth-first approach

H-Moivre finds the Hamming repetitions in a string s by constructing models of increasing length.

There are basically two ways of proceeding. The first one constructs the models in a breadth-first way, that is, as suggested above, it constructs first <u>all</u> the models of length 1, then <u>all</u> those of length 2, etc. At each step, the length of the models can be incremented by one or it can be doubled. Doubling the length of the models at each step gives a more time efficient algorithm but consumes a lot of memory since to construct the models of length $2k$, we need to store the information concerning all the models of length k.

The second way of proceeding, and the one we adopt, rests on the simple observation that to construct a model of length $k+1$, we only need the information concerning two very specific models, one of length k that is the prefix of the model we wish to construct, and the other of length 1 that is its suffix. If we store permanently the models of length 1, we can therefore construct any model m of length $k > 1$ in a depth-first way by storing the information about the models that are prefixes of m. This may result in a great economy of memory. The only difference in time complexity lies in the number of steps necessary to reach length k. This number is k in the depth-first approach, and $\log(k)$ in the breadth-first approach.

4.2.2 Time complexity of H-Moivre

Let s be a string of length n over Σ and let u be a word of length k of s starting at position i. There are at most :

$$N_{mod} = \sum_{i=0}^{e} \left(\begin{array}{c} k \\ i \end{array} \right) (\mid \Sigma \mid -1)^i$$

models of length k of which (i,d_s) can be an H-occurrence, for $k \geq e$. Since there are $n-k+1$ words of length k in s, the total number of elements in the stacks of P is thus at most $n.N_{mod}$. As each of these elements cannot be pushed into more than $\mid \Sigma \mid$ different stacks of Q, this results in no more than $\mid \Sigma \mid .n.N_{mod}$ operations to construct the models of length $k+1$. Majoring N_{mod}, H-Moivre's time complexity becomes :

$$O(n.k^{e+1}. \mid \Sigma \mid^{e+1})$$

where k is a fixed length or that of the longest repetition found.

4.2.3 Space complexity of H-Moivre

If we construct the models of H-Moivre in a depth-first way, its space complexity is at worst $O(n.k)$.

If we had chosen to construct these models in a breadth-first way, the space complexity of H-Moivre would be :

$$O(n.k^e. \mid \Sigma \mid^e).$$

4.2.4 Improving H-Moivre's performance

The governing parameter in H-Moivre's time complexity is thus e, the number of mismatches allowed. Our current implementation shows us that the algorithm is reasonably efficient for up to two or three mismatches. It should be observed however that the dependence on e is inherent to the problem and is not due to the algorithm. One can therefore improve the algorithm's performance by adding some constraints on the solutions. As it is now, H-Moivre finds all the M-Hamming repetitions, whatever the distribution of mismatches inside these repetitions. The algorithm can easily be modified so that the mismatches are evenly distributed.

This is done by specifying that every factor of length l of a word u has at most ϵ mismatches with its corresponding factor in m, where $\epsilon \leq e$.

Formally, we want that :

1. H-distance(u,m) $\leq e$;
2. $\forall\ i \in$ {1,...,|m|-l+1}, H-distance($u_i ... u_{i-l+1}$,$m_i ... m_{i-l+1}$) $\leq \epsilon$.

5 M-Levenshtein repetitions

5.1 Presentation of L-Moivre

The L-Moivre algorithm is defined in terms of the Levenshtein distance. Given parameters e and q and a string s, L-Moivre is able to solve the same problems as H-Moivre, except this time it finds models that are L-present in s.

Problems 1 and 2 are equivalent to finding all the sets $L(m)$ containing at least q elements with different positions, where $\mid m \mid$ is a fixed length k or is equal to k_{max}.

Although both algorithms work in the same way by doing special operations of set intersection to construct models of increasing length, there are important differences between them. The introduction of insertions/deletions leads to a more complicated algorithm, actually to several slightly different algorithms.

Before explaining how L-Moivre works, let us reexamine under another perspective the definition of an L-occurrence. Given a model m and one of its L-images u located at position i in s, an L-occurrence (i, d_s, d_d, d_i) is such that $d_s + d_d + d_i$ is a Levenshtein distance. In terms of 'dynamic programming', this means that if several paths exist aligning m with u, we only consider in $L(m)$ the one (or those in the case of equality) having the minimal sum.

Removing this condition gives the following definition :

Definition 5.1 *Given a model m and a word u starting at position i, (i, d_s, d_d, d_i) is an S-occurrence of m if u can be converted to m with d_s substitutions, d_d deletions and d_i insertions where $d_s + d_d + d_i \leq e$. u is called an S-image of m and we have :*

$$S(m) = \{(i, d_s, d_d, d_i) \mid d_s + d_d + d_i \leq e\}.$$

It is clear that $L(m) \subseteq S(m)$. This means that $S(m)$ contains optimal and non optimal paths. Note also that there are exactly the same number of different positions i in $S(m)$ and $L(m)$.

Given these definitions, L-Moivre uses the following lemmas to solve the problems proposed. Let s be a string of length n. Let m, m_1, m_2 be three models such that $m = m_1 m_2$ and $i \in$ {1,...,n-|m|+1} a position in s. Then :

Lemma 5.1

$$(i, d_s, d_d, d_i) \in S(m)$$

$$\Longrightarrow$$

$$\exists\, d_{s_1}, d_{d_1}, d_{i_1}, d_{s_2}, d_{d_2}, d_{i_2} \textit{ such that}$$

$$(i, d_{s_1}, d_{d_1}, d_{i_1}) \in S(m_1), (i+ \mid m_1 \mid -d_{d_1} + d_{i_1}, d_{s_2}, d_{d_2}, d_{i_2}) \in S(m_2)$$

$$\textit{and } d_s = d_{s_1} + d_{s_2},\ d_d = d_{d_1} + d_{d_2},\ d_i = d_{i_1} + d_{i_2}.$$

Lemma 5.2

$$(i, d_{s_1}, d_{d_1}, d_{i_1}) \in S(m_1),\ (i+ \mid m_1 \mid -d_{d_1} + d_{i_1}, d_{s_2}, d_{d_2}, d_{i_2}) \in S(m_2)$$

$$\textit{and } d_{s_1} + d_{d_1} + d_{i_1} + d_{s_2} + d_{d_2} + d_{i_2} \leq e$$

$$\Longrightarrow$$

$$(i, d_{s_1} + d_{s_2}, d_{d_1} + d_{d_2}, d_{i_1} + d_{i_2}) \in S(m_1 m_2 = m).$$

Note that lemma 5.1 is true with $L(m)$ instead of $S(m)$, but lemma 5.2 is not. This just means that the 'concatenation' of two optimal paths does not necessarily produce an optimal path. At least one concatenation does (from lemma 5.1), but not all of them. If we want to elaborate an algorithm based on a similar lemma with $L(m)$ instead of $S(m)$ then we must add an additional filtering step after each construction. We shall come back to it in a short while.

As in H-Moivre, the previous lemmas give a constructive property to build $S(m)$ from $S(m_1)$ and $S(m_2)$. As before, let us introduce a notation and a special operation of intersection on the sets of S-occurrences of two models m_1 and m_2.

Notation 5.1 *Given $S(m)$ a set of S-occurrences, we note :*

$$(S(m))_{-b} = \{(i,d_s,d_d,d_i) \mid (i+b,d_s,d_d,d_i) \in S(m)\}.$$

This is simply the set obtained from $S(m)$ by subtracting b to each i in $S(m)$.

Definition 5.2 *Given two sets of S-occurrences $S(m_1)$ and $S(m_2)$, we define the operation $\sqcap_{SL}$ by :*

$$\begin{aligned} S(m_1) \sqcap_{SL} S(m_2) = \{(i,d_s,d_d,d_i) \mid\ & (i,d_{s_1},d_{d_1},d_{i_1}) \in S(m_1), \\ & (i+d_{d_1}-d_{i_1},d_{s_2},d_{d_2},d_{i_2}) \in S(m_2), \\ & d_s = d_{s_1}+d_{s_2},\ d_d = d_{d_1}+d_{d_2},\ d_i = d_{i_1}+d_{i_2} \\ & \text{and } d_s+d_d+d_i \leq e\}. \end{aligned}$$

Then, from the lemmas, it comes :

Proposition 5.1 $L(m) \subseteq S(m) = S(m_1) \sqcap_{SL} S(m_2)_{-|m_1|}$.

Using property 5.1, the algorithm then proceeds in exactly the same iterative way as H-Moivre to find the models of increasing length that are S-present at least q times at different locations in s.

A major difference with H-Moivre however is that the $\sqcap_{SL}$ operation can bring into $S(m)$ several identical occurrences (i, d_s, d_d, d_i) of the model m. In the current implementation, whose complexity is discussed in sections 5.2.1 and 5.2.2, they are not filtered. Moreover, this operation can also bring into $S(m)$ several occurrences (i, d_s, d_d, d_i) that correspond to the same image u, that is, that have the same value $d_d - d_i$. It may appear more efficient not to keep them all. As mentioned before (lemma 5.1), keeping, for each length of the images u in $S(m)$, the one (or those) for which the distance is minimal is enough. This requires though an additional filtering step after each intersection is performed. The complexity of this variant is given in section 5.2.3.

The data structures used by L-Moivre are the same as in H-Moivre, that is, three arrays of stacks P, Q and V_k. However, what is now recorded in these stacks are the S-occurrences (i d_s,d_d,d_i).

5.2 Time and space complexities of L-Moivre

5.2.1 Time complexity of L-Moivre

Let s be a string of length n over Σ and let i be a position in s. There are at most :

$$N_{mod} = \sum_{j=0}^{e}\Big(\sum_{l=k-j}^{l=k+j}\Big(\sum_{\substack{d_i+d_d+d_s=j \\ l+d_i-d_d=k}} \binom{l}{d_i}\binom{l-d_i}{d_s} \mid \Sigma \mid^{d_s} A_l^{d_d} \mid \Sigma \mid^{d_d}\Big)\Big)$$

models of length k of which (i,d_s,d_d,d_i) can be an S-occurrence, for $k \geq e$, where $A_i^j = \frac{i!}{(i-j)!}$. Since there are $n-k+1$ words of length k in s, the total number of elements in the stacks of P is thus at most $n.N_{mod}$. As each of these elements cannot be pushed into more than $\mid \Sigma \mid$ different stacks of Q, this results in no more than $\mid \Sigma \mid .n.N_{mod}$ operations to construct the models of length $k+1$. Majoring N_{mod}, L-Moivre's time complexity becomes :

$$O(n.k^{2e+1}. \mid \Sigma \mid^{e+1})$$

where k is either a fixed length or that of the longest repetition found.

5.2.2 Space complexity of L-Moivre

If we construct the models of L-Moivre using the depth-first approach, its space complexity is at worst :

$$O(n.k^{2e+1}).$$

With the breadth-first approach, the space complexity is at most :

$$O(n.k^{2e}. \mid \Sigma \mid^{e}).$$

The difference between the two approaches in the case of L-Moivre is therefore less dramatic.

5.2.3 Improving L-Moivre's performance

As with H-Moivre, we can modify L-Moivre so that the errors are evenly distributed inside the repetitions.

Another way of improving L-Moivre's performance is, as suggested in section 5.1, to filter 'en route' $S(m)$ to obtain $L(m)$. That brings down L-Moivre's space complexity to $O(n.k.(e+1)^2)$ if the models are constructed in a depth-first way. Its worst-time complexity remains however the same. Whether it is advisable though, <u>in practice</u>, to perform such filtering at each step is still an open question. The answer depends greatly on the average-case time complexity of L-Moivre. We have not yet attempted to evaluate it. Let us observe also that the values given in this section correspond to the number of solutions that would be found if we were interested in finding <u>all</u> the models present in s, not just those present at least q times at different locations.

6 Conclusion and future work

We wish to emphasize the importance of the concept of model presented in this paper. Models have been introduced as an indirect means of comparing, in a multiple as opposed to a pairwise manner, all the words present in a set of strings. Although the problem of approximate multiple matching has great biological significance, few algorithms have addressed it without resorting to heuristics [7]. Models are also interesting in themselves since they are closely related to the concept of 'consensus word' which is widely used in the area of bio-computing.

The mechanism employed by KMR, and taken up by H and L-Moivre, provides just a more efficient way than the brute-force approach to construct the models. Other mechanisms can be imagined. The algorithms given here are reasonably efficient for up to two or three errors (substitutions and/or gaps) and, as previously mentioned, there is still room for further improvements.

Finally, it is easy to see that the notion of distance between objects can be generalized beyond that given by Hamming or Levenshtein : spatial distances can be used in the case of 3D structures or any scoring system in the case of sequences.

References

[1] R. Baeza-Yates and G. H. Gonnet. A new approach to text searching. *Commun. ACM*, 35:74–82, 1992.

[2] W. I. Chang and E. L. Lawler. Approximate string matching in sublinear expected time. pages 116–124. Proc. 31st Annu. Symp. on Foundations of Computer Science, 1990.

[3] Z. Galil and K. Park. An improved algorithm for approximate string matching. *SIAM J. on Computing*, 19:989–999, 1990.

[4] 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.

[5] G. M. Landau, U. Vishkin, and R. Nussinov. An efficient string matching algorithm with k differences for nucleotide and amino acid sequences. *Nucleic Acids Res.*, 14:31–46, 1986.

[6] A. M. Landraud, J. F. Avril, and P. Chretienne. An algorithm for finding a common structure shared by a family of strings. *IEEE Transactions on pattern analysis and machine intelligence*, 11(8):890–895, 1989.

[7] M.-Y. Leung, B. E. Blaisdell, C. Burge, and S. Karlin. An efficient algorithm for identifying matches with errors in multiple long molecular sequences. *J. Mol. Biol.*, 221:1367–1378, 1991.

[8] M. F. Sagot, A. Viari, J. Pothier, and H. Soldano. Finding flexible patterns in a text - an application to 3D molecular matching. pages 117–145, Seattle, Washington, USA, 1994. First International IEEE Workshop on Shape and Pattern Matching in Computational Biology.

[9] H. Soldano, A. Viari, and M. Champesme. Searching for flexible repeated patterns using a non transitive similarity relation. *Pattern Recognition Letters, in press*, 1994.

[10] J. Tarhio and E. Ukkonen. Boyer-Moore approach to approximate string matching. In *Lecture Notes in Computer Science*, volume 447, pages 348–359. Springer-Verlag, 1990.

[11] E. Ukkonen. Algorithms for approximate string matching. *Inform. Contr.*, 64:100–118, 1985.

[12] E. Ukkonen. Finding approximate patterns in strings. *J. Algorithms*, 6:132–137, 1985.

[13] S. Wu and U. Manber. Agrep - a fast approximate pattern-matching tool. pages 153–162, San Francisco, CA, 1992. USENIX Technical Conference.

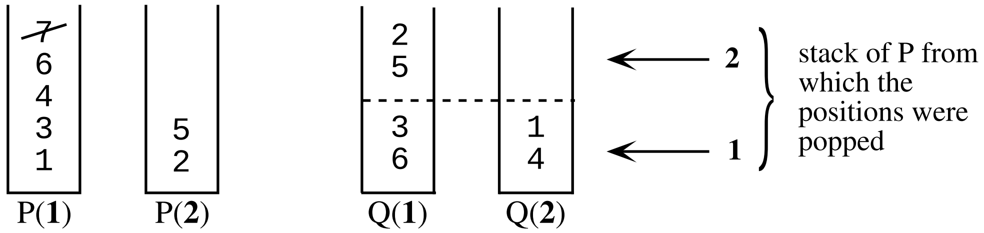
[14] S. Wu and U. Manber. Fast text searching allowing errors. *Commun. ACM*, 35:83–91, 1992.

$\Sigma = \{a, b\}$, $q = 2$

s =	a	b	a	a	b	a	a
positions	1	2	3	4	5	6	7
V_1 =	**1**	**2**	**1**	**1**	**2**	**1**	**1**

There are 2 classes of words of length 1 :
a : **1**
b : **2**

Construction of the classes of words of length 2 : use of two arrays of stacks, P and Q.
- For each position p, push p into $P[V_1[p]]$.
- Pop p from each P[i], and push p into $Q[V_1[p+1]]$ (if $p+1 \leq n$).

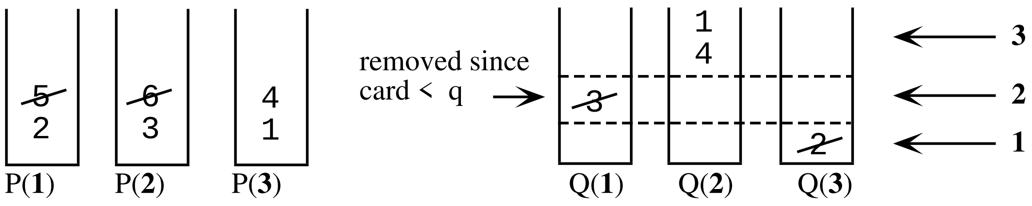


$c \leftarrow 1$
Pop each p from Q[i], and push c into $V_2[p]$. Increment c each time a new Q stack is started, or a dashed line is crossed.

	1	2	3	4	5	6	7
V_2 =	**3**	**1**	**2**	**3**	**1**	**2**	

There are 3 classes of words of length 2 : ba : **1**, aa : **2**, ab : **3**.

Construction of the classes of words of length 4.



	1	2	3	4	5	6	7
V_4 =	**1**			**1**			

There is 1 class of (repeated) words of length 4 : abaa : **1**.

Figure 1: An example of the KMR algorithm.

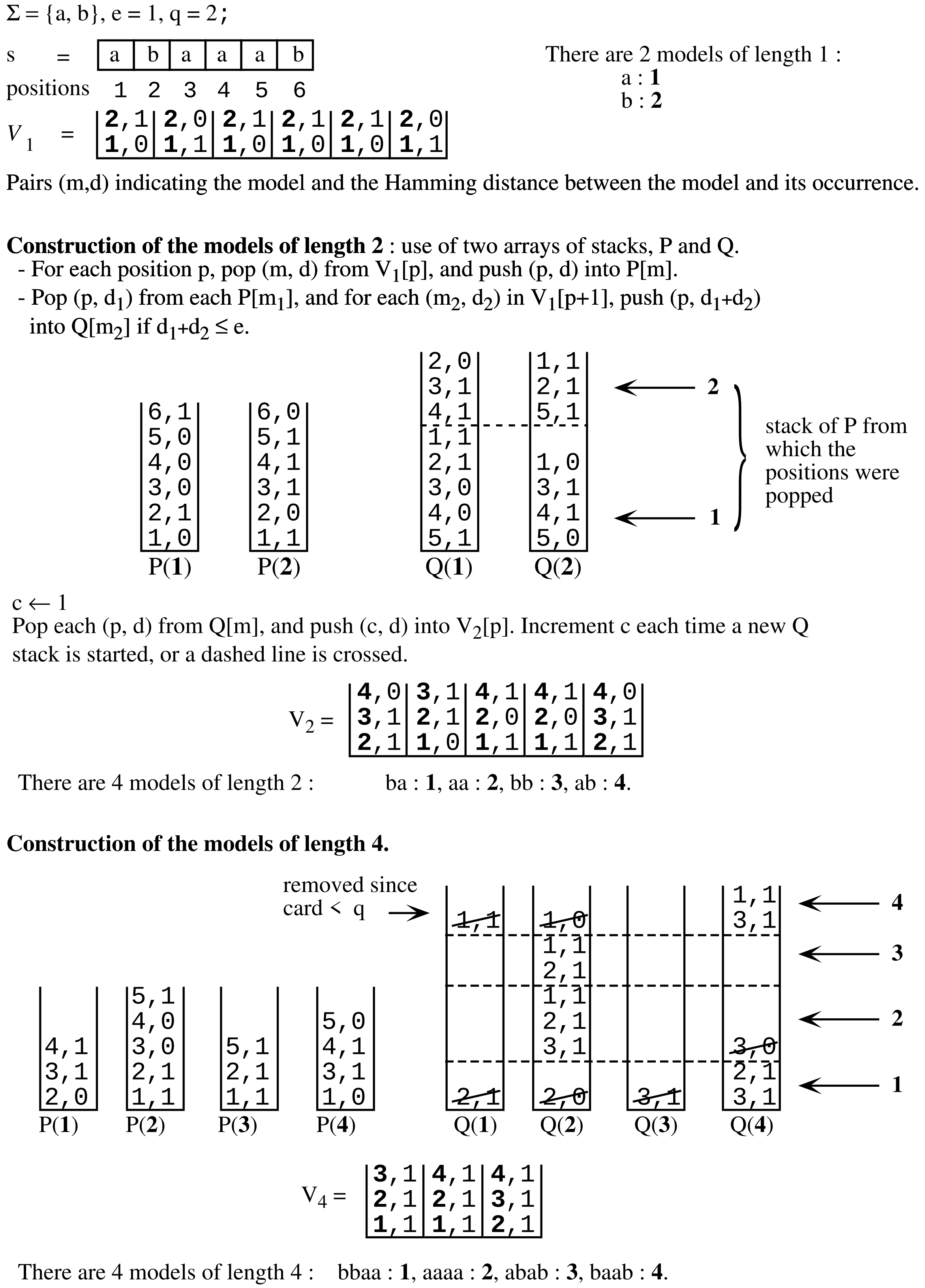


Figure 2: An example of the H-Moivre algorithm.