ELSEVIER

March 1995

Pattern Recognition Letters 16 (1995) 233-246

Pattern Recognition Letters

Searching for flexible repeated patterns using a non-transitive similarity relation

Henri Soldano [a,c,*], Alain Viari [b,c], Marc Champesme [a,c]

[a] *Laboratoire d'Informatique de Paris Nord, URA-CNRS 1507, Institut Galilée, Université Paris Nord, Av. J.B. Clément, 93430 Villetaneuse, France*

[b] *Laboratoire de Physique et Chimie Biomoléculaires, URA-CNRS 198, Institut Galilée, Université Paris Nord, Av. J.B. Clément, 93430 Villetaneuse, France*

[c] *Atelier de Bio-Informatique, section Physique-Chimie, Institut Curie, 11 rue Pierre et Marie Curie, 75005 Paris, France*

Received 15 May 1993; revised 2 August 1994

Abstract

Given a reflexive and symmetric, but not necessarily transitive, similarity relation defined on an alphabet of symbols, two objects of size k are related if, at each position, their symbols are related. Then, given a set of objects, we are interested in maximal subsets of related objects. We give some general properties of these subsets and we propose algorithms for identifying them in the particular case of k-length substrings in a string. These algorithms derive from the Karp, Miller and Rosenberg algorithms for the identification of repeated patterns.

Introduction

In a previous work, Karp et al. (1972) have proposed various algorithms, hereafter referred to as KMR algorithms, to identify repeated patterns in a structure of size N (string, array or tree). In their approach the patterns to identify correspond to exact matches between objects. For instance, two k-length substrings match if, at each position, the same symbol is present in both substrings. However some situations require a more flexible matching and patterns corresponding to similar, rather than strictly identical, objects are searched for. As an example, in molecular biology similar fragments of amino acid sequences may exhibit a similar 3D-structure or biological function. In this work we suppose that a symmetric and reflexive relation R, expressing a similarity, is defined on the alphabet. This leads to a reflexive and symmetric relation R_k on structured objects of size k, defined as follows: two objects are similar if, at each position, the corresponding symbols are related by R. Now, given a set of objects, our purpose is to find all the maximal cliques of the relation R_k, i.e., the maximal subsets in which any pair of objects (x, y) is such that x and y are related. Then each maximal clique of R_k defines a pattern.

The remainder of this paper is divided into four sections. Section 1 presents some definitions and properties concerning the maximal cliques of R_k. This includes some constructive properties for structured objects of size k.

* Corresponding author.

Section 2 presents some algorithms, implementation, and experiments on random strings addressing the following problems: (1) find all maximal cliques of R_k, whose size is greater than 2 (i.e., repeated patterns) when the objects are the k-length substrings of an N-length string; (2) find the largest k for which such repeated patterns exist in an N-length string. Experiments are discussed in Section 3 and a conclusion is given in Section 4.

1. Properties of the relation R_k

1.1. Notations

Throughout this paper, we will use the following notations:
$A = \{\sigma_1, \ldots, \sigma_s\}$ is an alphabet of s symbols.
R is a symmetric and reflexive (but not necessarily transitive) relation, defined on A, which represents a similarity between the symbols of A.

An "object" of size k is a labeled and oriented structure composed of k elements of A. Each element is assigned to a certain "position".

For instance, a k-length string is a structure of size k, and an $l \times c$ array is a structure of size $k = l \cdot c$.

It is important to distinguish between the name of an object and its "value" (i.e., the components of its structure). More precisely, if x is an object of size k, we will note its value as

$$V(x) = (V_1(x), V_2(x), \ldots, V_k(x)), \quad \text{where } V_i(x) \text{ belongs to } A.$$

For example, consider a string $S = (s_1, s_2, \ldots, s_N)$ of length N. The set $X = \{1, 2, \ldots, y, \ldots, N-k+1\}$ of positions in the string is a set of $N-k+1$ objects representing all the k-length substrings of S. An object (substring) of size k will be referred to as j (its starting index) and its value is $V(j) = (s_j, s_{j+1}, \ldots, s_{j+k-1})$.

Unless specified, we will denote by X the set of all objects of size k we are considering and by N the size of X.

1.2. Definitions

The comparison between two objects of the same size will result from the comparisons of symbols at each position. To this purpose, we define the following relation:

Definition 1. R_k stands for the symmetric and reflexive relation on X, defined as

$$x\, R_k\, y \quad \Leftrightarrow \quad \forall i \in \{1, 2, \ldots, k\}, \quad V_i(x)\, R\, V_i(y).$$

A *clique* of the relation R_k on X is a subset of X in which any pair of elements (x, y) is such that $x\, R_k\, y$. A clique is called *maximal* if by adding any other element to it, the resulting subset is no longer a clique.

Hereafter, we denote by C_X a clique of the relation R_k on X.

In the same manner, one can consider the cliques of the relation R on A. To distinguish them from the cliques of R_k, we will use a lowercase c for such cliques and an uppercase C for the cliques of R_k.

Let $\{c_1, c_2, \ldots, c_k\}$ be a set of k cliques of R. We will call the Cartesian product $c = c_1 \times c_2 \times \cdots \times c_k$ a *clique product* of R. Moreover, if all the c_i are maximal cliques, then c will be called a *maximal clique product.*

We will use the following example throughout the paper:

Let A = $\{a, b, c, d, e\}$. R is a symmetric and reflexive relation whose graph is represented on Fig. 1. The maximal cliques of R are the sets

$$c_1 = \{a, b, c\}, \qquad c_2 = \{b, c, d\}, \qquad c_3 = \{d, e\}, \qquad c_4 = \{a, e\}.$$

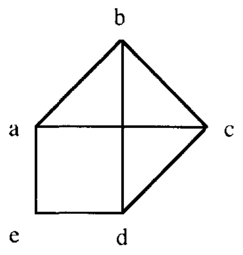


Fig. 1. Graph of a symmetric and reflexive relation. The edges due to reflexivity are omitted.

Let us consider a string $S = adbeb$, and the set $X = \{1, 2, 3\}$ of the 3-length substrings of S represented by their starting indexes. Then we have, for instance, $1\ R_3\ 3$ since $a\ R\ b$, $d\ R\ e$, and $b\ R\ b$. Moreover $\{1, 3\}$ is a maximal clique of R_3 since it cannot be extended.

1.3. Properties of R_k

We now give several properties of the maximal cliques of R_k.

Let f be the function that relates any clique C_X of R_k to the Cartesian product $e_1 \times e_2 \times \cdots \times e_k$, where each e_i stands for the subset of A defined as follows:

Definition 2. $e_i = \bigcup_{x \in C_X} V_i(x)$.

Proposition 1. *$f(C_X)$ is a clique product of R.*

Proof. For each position i, the values of V_i corresponding to the elements of C_X are related by R (Definition 1). As a consequence e_i is a clique of R. □

Proposition 2. *Given a maximal clique product $\bar{c}$ of R, there exists at most one maximal clique C_X of R_k such that $f(C_X) \subseteq \bar{c}$.*

Proof. Let C_X and C'_X be two maximal cliques of R_k, such that $f(C_X) \subseteq \bar{c}$ and $f(C'_X) \subseteq \bar{c}$. Then *for each position i*, we obtain (Definition 2):

$$e_i \cup e'_i = \bigcup_{x \in C_X \cup C'_X} V_i(x) \subseteq \bar{c}_i .$$

This means that $C_X \cup C'_X$ is a clique of R_k, and, since C_X and C'_X are both maximal cliques, this necessarily implies that $C_X = C'_X$. □

From these propositions one derives the following:

Proposition 3. *Let $\underline{f}$ be the restriction of f to the set of maximal cliques of R_k. Then $\underline{f}$ is one to one.*

Proof. Let C_X and C'_X be two maximal cliques of R_k such that $f(C_X) = f(C'_X)$. Since $f(C_X)$ is a clique product (Proposition 1), there exists at least one maximal clique product $\bar{c}$ such that $f(C_X) = f(C'_X) \subseteq \bar{c}$. Then it follows (Proposition 2) that $C_X = C'_X$. □

Proposition 4. *Given*:
e_k: The number of maximal cliques of R_k.

m: The number of maximal cliques of R.
g: The maximum number of maximal cliques of R containing the same symbol of A.
Γ_k: The set of maximal cliques of R_k.
We have the following upper bounds:
(a) $\sum_{C_X \in \Gamma_k} \mathrm{Card}(C_X) \leqslant N \cdot g^k$;
(b) $e_k \leqslant \min(N \cdot g^k, m^k)$.

Proof. (a): By definition of g, each element x of X is such that $V(x)$ belongs to less than g^k maximal clique products. Then, according to Proposition 2, x belongs to less than g^k maximal cliques of R_k. Since N is the size of X, the inequality is proved.

(b): The number of maximal clique products is equal to m^k. Then, according to Proposition 2, we have $e_k \leqslant m^k$. Furthermore, since each maximal clique of R_k contains at least one element, it follows from (a) that $e_k \leqslant N \cdot g^k$. □

In the particular case where R is an equivalence relation, $g=1$ and these inequalities simply mean that the classes of R_k form a partition of X, and that the number of classes is less than $\min(N, m^k)$. The number g, hereafter referred to as the "degeneracy" of R, represents a discrepancy from R to an equivalence relation. One could also consider, as a more realistic measure in practical cases, the averaged degeneracy $\bar{g}$, obtained by assigning an a priori probability to each symbol. Then, as an example, for a random N-length string the averaged quantities in Proposition 4(a)–(b) become of order $O(N \cdot \bar{g}^k)$.

Proposition 5. *Let C_X be a clique of R_k, and C'_X be a maximal clique of R_k. Then we have*

$$C_X \subseteq C'_X \quad \Leftrightarrow \quad f(C_X) \subseteq f(C'_X) .$$

Proof. $\Rightarrow$: Follows from the definition of f.

$\Leftarrow$: Let x be an element of C_X. Then, by definition of f, $V(x) \in f(C_X)$ and thus $V(x) \in f(C'_X)$. In the same manner, *for any element x'* of C'_x we have $V(X') \in f(C'_X)$. Hence $V(x)$ and $V(x')$ belong to the same clique product $f(C'_X)$. It follows from Definition 1 that $x \; R_k \; x'$. Since C'_X is a maximal clique of R_k, x must belong to C'_X, and then $C_X \subseteq C'_X$. □

Hence, f appears as a "characteristic function" of cliques of R_k. This allows us to compare maximal cliques (Proposition 3), and to check whether a clique is included into a given maximal clique (Proposition 5) in a constant number of operations with respect to N. Furthermore we give upper bounds both for the total number of objects in the whole set of maximal cliques of R_k, and for the size e_k of this set. These bounds are linear with respect to N, and depend on the degeneracy g of R.

The previous properties are quite general in the sense that they do not depend upon a particular structure but only upon its size. We will now focus on other properties allowing step by step constructions of maximal cliques of R_k, for particular structures.

1.4. Constructing maximal cliques of R_k for k-length substrings of an N-length string

In a previous work, Karp, Miller and Rosenberg (KMR) (1972) have addressed the problem of identifying repeated substructures of fixed size in a structure of size N (string, array or tree). In their work, the relation R between the symbols was the identity. In the case of repeated substrings in an N-length string, the algorithms they proposed were based on two lemmas:

Lemma 1 (KMR, 1972). *$x \, R_{a+b} \, y \quad \Leftrightarrow \quad x \, R_a \, y$ and $x+b \, R_a \, y+b$ (with $b \leqslant a$).*

Lemma 2 (KMR, 1972). *$x\, R_{a+b}\, y \Leftrightarrow x\, R_a\, y$ and $x+a\, R_b\, y+a$.*

These lemmas allow the construction of the R_{a+b} relation starting from the R_a (and R_b in Lemma 2) relation. When R is not transitive, these lemmas still hold (since they only involve pairwise comparisons).

1.4.1. Constructing maximal cliques of R_1

In the case of an N-length string $S=(s_1, s_2, \ldots, s_N)$, and considering its 1-length substrings, X is the set $\{1, 2, \ldots, y, \ldots, N)$ of all positions in S (see Section 1.1).

Let us denote by:

$\{\bar{c}_1, \bar{c}_2, \ldots, \bar{c}_m\}$: The m maximal cliques of R.

$\{C_X^1, C_X^2, \ldots, C_X^m\}$: The subsets of X defined as $C_X^i=\{j\in X \mid s_j \in \bar{c}_i\}$.

Then we have the following properties:

Proposition 6. (a) *C_X^i is a clique of R_1.*
(b) *The set of maximal cliques of R_1 is included in $\{C_X^1, C_X^2, \ldots, C_X^m\}$.*

Proof. (a): For any x, y in C_X^i, we have $s_x\, R\, s_y$ and thus $x\, R_1\, y$.

(b): Let C be a maximal clique of R_1. Then $c=f(C)$ is a clique of R (Proposition 1). Thus, there exists a maximal clique $\bar{c}_i$ of R, such that $c \subseteq \bar{c}_i$. Then, since C_X^i contains all the elements j such that $s_j \in \bar{c}_i$, it follows that $C \subseteq C_X^i$. But, since C is a maximal clique of R_1, this necessarily implies that $C=C_X^i$. □

Hence, the set of maximal cliques of R_1 is obtained by constructing $\{C_X^1, C_X^2, \ldots, C_X^m\}$, and by eliminating all the C_X^i which are included in any other C_X^j.

In our previous example (Section 1.2), we have with $S=adbeb$:

$$C_X^1=\{1,3,5\}, \qquad C_X^2=\{2,3,5\}, \qquad C_X^3=\{2,4\}, \qquad C_X^4=\{1,4\}.$$

Note that in this example all the C_X^i are maximal.

1.4.2. Step-by-step construction of maximal cliques of R_k

Given a set E of indices, let us denote by E_{+d} (resp. E_{-d}) the set obtained by adding (resp. subtracting) the integer d to each index in E. Consider a clique C_X of R_a. Then any pair $\{x, y\}$ belonging to $(C_X)_{-b}$ is such that $x+b\, R_a\, y+b$.

The following properties are established from the KMR lemmas:

Proposition 7. (a) *Let C_X be a clique of R_a and C'_X be a clique of R_a. Then $C_X \cap (C'_X)_{-b}$ is a clique of R_{a+b}.*
(b) *Let C_X be a clique of R_a and C'_X be a clique of R_b. Then $C_X \cap (C'_X)_{-a}$ is a clique of R_{a+b}.*

Proof. (a): For any pair $\{x, y\} \subseteq C_X \cap (C'_X)_{-b}$ we have $x\, R_a\, y$ since $\{x, y\} \subseteq C_X$, and $x+b\, R_a\, y+b$ since $\{x, y\} \subseteq (C'_X)_{-b}$. Then, according to Lemma 1, it follows that $x\, R_{a+b}\, y$. As a consequence $C_X \cap (C'_X)_{-b}$ is a clique of R_{a+b}.

(b): Proof is similar. □

Proposition 8. *Let C''_X be a maximal clique of R_{a+b}. Then:*
(a) *There exist a maximal clique C_X of R_a and a maximal clique C'_X of R_a such that $C''_X = C_X \cap (C'_X)_{-b}$.*
(b) *There exist a maximal clique C_X of R_a and a maximal clique C'_X of R_b such that $C_X \cap (C'_X)_{-a}$.*

Proof. (a): For any pair $\{x, y\} \subseteq C''_X$, we have $x\, R_{a+b}\, y$ and thus, according to Lemma 1, we have $x\, R_a\, y$ and $x+b\, R_a\, y+b$.

From $x\, R_a\, y$, it follows that C''_X is a clique of R_a and thus there exists a maximal clique C_X of R_a such that $C''_X \subseteq C_X$. From $x+b\; R_a\; y+b$ it follows that $(C''_X)_{+b}$ is a clique of R_a and thus there exists a maximal clique C'_X of R_a such that $(C''_X)_{+b} \subseteq C'_X$ and therefore $C''_X \subseteq (C'_X)_{-b}$.

As a consequence we have $C''_X \subseteq C_X \cap (C'_X)_{-b}$. However, according to Proposition 7(a), $C_X \cap (C'_X)_{-b}$ is a clique of R_{a+b} and since C''_X is supposed to be maximal, the inclusion necessarily implies that $C''_X = C_X \cap (C'_X)_{-b}$.

(b): Proof is similar. □

Starting from the set of maximal cliques of R_a (resp. R_a and R_b), we will obtain all the maximal cliques of R_{a+b} by performing all the intersections mentioned above. Some intersections will produce non-maximal cliques which can be eliminated by checking if there exists any other clique which includes them.

Let us consider again our previous example (Section 1.2).

When $a=b=1$, the two lemmas are equivalent, and we will use the maximal cliques C_X of R_1: $\{\{1,3,5\}, \{2,3,5\}, \{2,4\}, \{1,4\}\}$, together with their corresponding $(C_X)_{-1}$: $\{\{0,2,4\}, \{1,2,4\}, \{1,3\}, \{0,3\}\}$. Performing all the intersections and removing non-maximal cliques yields $\{\{2,4\}, \{1,4\}, \{1,3\}\}$ as the maximal cliques of R_2. They correspond to the substrings $\{\{db, eb\}, \{ad, eb\}, \{ad, be\}\}$.

This clearly leads to a step-by-step algorithm which takes as input an N-length string and a relation R and which produces all maximal cliques of R_k.

2. Algorithms, implementation and experiments

2.1. Algorithms

In this section, we address the problem of finding all the repeated k-length substrings in a string, according to a relation R between symbols. In the previous section, we have proposed a method to construct all maximal cliques of R_k in an N-length string. This clearly solves the problem since the repeated k-length substrings are the maximal cliques of R_k whose size is larger than two. The repeated patterns are constructed by using the previous method except that, at the end of each step, the maximal cliques whose size is less than two are removed. This strategy is justified since if a given maximal clique C_X of R_k is of size less than two, then its intersection with any set will also be of size less than two and thus will not produce a longer repeated substring.

The first algorithm solves the following problem:

Problem 1. Given an alphabet A, a symmetric and reflexive relation R, and a string S, find all its repeated substrings of length k.

This problem generalizes, by using a similarity relation R rather than the identity, a problem solved by KMR.

Algorithm 1

```
Step 1. Construct the set of maximal cliques of $R$.
Step 2. Construct the set $L_1$ of repeated substrings of length 1 as seen in Section 1.4.1 except that all cliques of size
        less than two are removed before removing non-maximal cliques of $R_1$.
    Initialize $i$ to 1.
Step 3. Repeat while $2i \leqslant k$.
    Construct the set $L_{2i}$ of repeated substrings of length $2i$, by using $L_i$ as seen in Section 1.4.2:
      Lemma 1 is used with $a=b=i$. The intersections mentioned in Proposition 7(a) concern only the
      maximal cliques of $R_i$ belonging to $L_i$.
```

```
        Remove the resulting cliques of $R_{2i}$ whose size is less than two.
        Remove non-maximal cliques by using an inclusion test.
        Multiply $i$ by 2.
Step 4. If $k = 2i$.
        return $L_{2i}$ as result.
      else
        construct $L_k$ as in Step 3 by using Lemma 1 with $a = 2i$, $b = k - 2i$.
        return $L_k$.
```

The second algorithm solves the following problem:

Problem 2. With the same inputs as in Problem 1, find the largest integer $k_{\max}$ such that there exists at least one repeated substring of length $k_{\max}$, and return this substring(s).

As above, this generalizes a problem solved by KMR.

Algorithm 2

```
Step 1. Same Step 1 as in Algorithm 1.
Step 2. Same Step 2 as in Algorithm 1.
Step 3. Repeat while $L_i$ is not empty
        same operations as Step 3 in Algorithm 1
Step 4. Perform a binary search by using Lemma 1. Namely: use $L_{i/2}$ to construct $L_{3i/4}$. If this latter is not empty
        then use it to construct $L_{7i/8}$ else use $L_{i/2}$ to construct $L_{5i/8}$, and so on.
```

2.2. Implementation

Because of the similarity of the following implementation to those described in (Karp et al., 1972), we have followed the same notations whenever possible.

At a given step, say that R_a has e_a maximal cliques labeled from 1 to e_a. The occurrence of these cliques on the string is represented as an $N-a+1$ place vector of lists $v_a = (v_a(1), v_a(2), \ldots, v_a(N-a+1))$, where each $v_a(i)$ is the list of all the labels of the maximal cliques of R_a to which position i belongs.

These lists are actually implemented as pushdown stacks with the traditional "Push" and "Pop" operations. However at some steps, we need to read down the content of a stack without actually popping the values. For this purpose, we provide the stack with an additional "Downread" operation.

Now we have to construct R_{a+b} from R_a, by using Lemma 1 and the above representation. Assume that we have at our disposal the vector v_a stored as indicated above, and two initially empty e_a place vectors of stacks available. Call them $P = (P(1), P(2), \ldots, P(e_a))$ and $Q = (Q(1), Q(2), \ldots, Q(e_a))$.

2.2.1. Pseudocode

```
// Step 1. (Build P)
For $i = 1$ to $N - a + 1$
  While $v_a(i)$ is not exhausted
        c ← Downread($v_a(i)$)
        Push $i$ into $P(c)$
  EndWhile
EndFor
```

//P gives an explicit representation of the maximal cliques of R_a, one maximal clique in each $P(c)$. This is the dual //representation of v_k which gives, for each position, the labels of the maximal cliques at this position.

```
// Step 2. (Build Q)
For $j=1$ to $e_a$
  While $P(j)$ is not empty
      $s \leftarrow \mathrm{Pop}(P(j))$
      If $s+b \leqslant N-a+1$ then
        While $v_a(s+b)$ is not exhausted
            $c \leftarrow \mathrm{Downread}(v_a(s+b))$
            Push $s$ into $Q(c)$
            Push $j$ into $Q(c)$          //also push P-origin into Q
        EndWhile
      EndIf
  EndWhile
End For
```

//This gives us the maximal cliques of R_a shifted so that all pairs of integers (x, y) occurring on the same Q-stack //are such that $x+b\ R_a\ y+b$. This means that, to a given $Q(c)$, there corresponds a maximal clique C' of R_a, such //that $Q(c)$ represents the set $(C')_{-b}$, mentioned in Section 1.4.2. The reason why j (the label of the P-stack from //which s come(s) is also pushed into Q will appear at the next step.

```
// Step 3. (Construct $v_{a+b}$)
$e_{a+b} \leftarrow 0$                   //a clique counter
For $j=1$ to $e_a$
  previous $\leftarrow 0$               //initialized to a dummy P label
  While $Q(j)$ is not empty
      $c \leftarrow \mathrm{Pop}(Q(j))$        //the P label of s
      $s \leftarrow \mathrm{Pop}(Q(j))$
      If $c \neq$ previous
        $e_{a+b} \leftarrow e_{a+b}+1$    //start a new clique of $R_{a+b}$
        previous $= c$
      EndIf
      Push $e_{a+b}$ into $v_{a+b}(s)$
  EndWhile
EndFor
```

//v_{a+b} is a representation of the set of cliques of R_{a+b} obtained by performing the intersections mentioned in //Proposition 7(a). Note that e_{a+b} is incremented each time a new $Q(j)$ is considered or each time a new value //of c appears. It should be pointed out that, since only repeated patterns are searched for rather than all maximal //cliques, the previous pseudocode should be modified so that only the labels corresponding to cliques whose size //is greater than of equal to two are actually pushed into v_{a+b}.

```
// Step 4. (Remove non-maximal cliques of $R_{a+b}$)
//First construct a vector $T=T(1), \ldots, T(e_{a+b})$ of $e_{a+b}$ stacks. Each T-stack will contain one of the cliques of
//$R_{a+b}$ represented in $v_{a+b}$. In addition, the status of each T-stack is flagged as "+", "−" or "?". "+" means
//"clique is maximal"; "−" means "clique is not maximal" and "?" means "status of clique is unknown".
Build $T$ from $v_{a+b}$ as for $P$ in Step 1.
Mark all $T$-stacks as "?".
For $i=1$ to $N-(a+b)+1$
  // We will fix once and for all the status of all T-stacks at position i.
```

```
 While $v_{a+b}(i)$ is not empty
   $c \leftarrow$ Pop($v_{a+b}(i)$)
   If $T(c)$ is not marked as "–"                          // T(c) could be maximal.
     // Check T(c) against all remaining T-stacks.
     While $v_{a+b}(i)$ is not exhausted and $T(c)$ is not marked as "–"
       d $\leftarrow$ DownRead($v_{a+b}(i)$)
       If $T(d)$ is not marked as "–"                      // T(d) could be maximal
         If $T(c)$ is marked as "?"                        // T(c) status still unknown
           If $T(c) \subseteq T(d)$ then mark $T(c)$ as "–" EndIf
         EndIf
         If $T(d)$ is marked as "?"                        // T(d) status still unknown
           If $T(d) \subseteq T(c)$ then mark $T(d)$ as "–" EndIf
         EndIf
       EndIf
     EndWhile
     // Now the final status of T(c) is known: if it is not included in any T(d) at this position then it is maximal.
     If $T(c)$ is not marked as "–" then mark it as "+" EndIf
 EndWhile
EndFor
// Now the status of each T-stack is determined.
Rebuild $v_{a+b}$ from $T$ by using only the T-stacks marked as "+"
// End
```

2.2.2. Complexity

Now, we will consider the complexity of each separate step.

Step 1: At the beginning of Step 1, we have at most $N \cdot g^a$ indices in v_a (Proposition 4(a)). Thus, Step 1 requires $O(N \cdot g^a)$ operations.

Step 2: This step requires $O(N \cdot g^{a+b})$ operations. The proof of this bound is more difficult. Let us consider, for the sake of simplicity, the cases of Lemma 2 or the case of Lemma 1 with $a=b$ (which is similar), since these are the cases mostly used in Algorithms 1 and 2 above. Let us now consider a particular index s popped from $P(i)$ and its associated a-length substring at position s. Each symbol of this substring cannot belong to more than g maximal cliques of R (by definition of g), thus, this a-length substring cannot belong to more than g^a maximal clique products and, hence, to more than g^a maximal cliques of R_a (this follows from Proposition 3). Therefore, the index s cannot appear more than g^a times in the P-stacks. Each time s is popped from a P-stack, it will be pushed in the Q-stacks indicated in $v_b(s+a)$. Thus, let us consider, in the same manner, the b-length word at the position $s+a$. The same argument as above shows that this word cannot belong to more than g^b maximal cliques of R_b and, hence, s cannot be pushed into more than g^b Q-stacks. Thus, a given index s cannot appear more than $g^a \cdot g^b = g^{a+b}$ times in all Q-stacks. Finally, there are at most $N \cdot g^{a+b}$ elements in all the Q-stacks and the whole step has required $O(N \cdot g^{a+b})$ operations. Analogous arguments with Lemma 1 and $a \neq b$ would lead to $O(N \cdot g^{2a})$ instead of $O(N \cdot g^{a+b})$.

Step 3: This step requires the same number of operations as Step 2 since at most $O(N \cdot g^{a+b})$ indices have been pushed into Q-stacks, as previously stated. Note that, since a clique contains at least one element (actually 2 for repeated substrings), $e_{a+b} \leqslant N \cdot g^{a+b}$.

Step 4: The construction of the T-stacks requires $O(N \cdot g^{a+b})$ operations (this is the same operation as in Step 1 but now with at most $N \cdot g^{a+b}$ elements).

Then we perform all necessary inclusion tests by using v_{a+b} and the T-stacks. First, remember that we have previously stated that each index s cannot appear more than g^{a+b} times in the Q-stacks and thus each $v_{a+b}(d)$ cannot contain more than g^{a+b} cliques. Therefore, we have to perform at most $O(N \cdot (g^{a+b})^2)$ inclusion tests.

Finally, we need to evaluate how many operations are required by one inclusion test. This depends on how the inclusion test (between two T-stacks representing two cliques) is conducted: there are basically two ways to do this. The easiest way is to compare each position indicated in the two T-stacks. Since the indices are ordered in each clique, this will take at most $O(N)$ operations. This first method would thus lead to $O(N^2 \cdot (g^{a+b})^2)$ operations for the overall step. There is, however, another way to perform the inclusion test. Remember that Proposition 5 states that any inclusion test on cliques of R_k (here $k=a+b$) can be performed on clique products. Thus, if all the clique products are assumed to be known, one inclusion test would thus require $O(k)$ operations (considering Card(A) as constant). The explicit representation of all the c_f clique products can be conducted separately and requires $O(k \cdot N \cdot g^k)$ (this follows from Definition 2). Finally, this second method leads to $O(N \cdot k \cdot g^{2k})$ for the overall step.

Note, however, that for most practical cases (i.e., with non biased strings), the first method is linear with respect to N (this comes from the fact that when many cliques have to be considered their size is actually very small) and is simpler. Note also that for $g=1$ there is only 1 clique (class) in each $v_{a+b}(d)$ and hence no inclusion test at all is performed. Hence the algorithm behaves, in this particular case, like the original KMR algorithm.

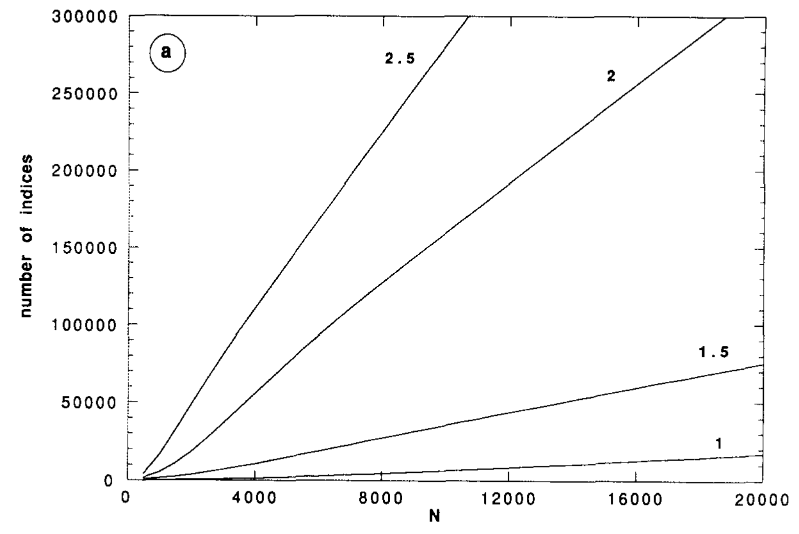
Finally, whole worst-case complexity of Steps 1 to 4, i.e., of construction of maximal cliques of R_k, is of order $O(N \cdot k \cdot g^{2k})$ and thus the overall algorithm is of order $O(N \cdot k_{\max} \cdot g_{\max}^{2k}) \cdot \log(k_{\max})$ ($k_{\max}$ being either a chosen length (Problem 1) or the maximum possible length (Problem 2)).

Now we present some experimental results obtained on random sequences.

2.3. Experiments

The following experiments concern Algorithm 1. The program is written in C and is run on a SUN Sparcstation (28 Mips). We present two sets of experiments, corresponding to two alphabets of respectively 10 (experiment 1) and 26 (experiment 2) latin letters. In both cases, we are searching for repeated patterns of length $k=4$, using four different relations R between symbols. The first relation is the identity, i.e., has a degeneracy $g=1$, in that case the algorithm is equivalent to the original KMR algorithm. The three other relations R are designed to have averaged degeneracies $\bar{g}$ equal to 1.5, 2, and 2.5, and numbers of maximal cliques $m=10$ (experiment 1) and $m=26$ (experiment 2). This is obtained by using the following circular relation R: in experiments with $\bar{g}=2$, the maximal cliques of $\bar{R}$ are $\{\{a, b\}, \{b, c\}, \ldots, \{i, j\}, \{j, a\}\}$ (for experiment 1) and $\{\{a, b\}, \{b, c\}, \ldots, \{y, z\}, \{z, a\}\}$ (for experiment 2); in experiments with $\bar{g}=1.5$, half of the previous maximal cliques contains two elements as before and the other half contains only one element (e.g. $\{\{a, b\}, \{b, c\}, \ldots, \{m, a\}, \{n\}, \ldots, \{x\}, \{y\}, \{z\}\}$ for experiment 2); in experiments with $\bar{g}=2.5$, half of the maximal cliques contains two elements and the others contain three elements (e.g. $\{\{a, b, c\}, \{b, c, d\}, \ldots, \{l, m, a\}, \{m, a, b\}, \{n, o\}, \ldots, \{y, z\}, \{z, n\}\}$ for experiment 2). For each of these cases, we built a random string of size N increasing from 500 to 20 000 by step $\Delta N=500$. For each N, we record the CPU time (t) and the total number of indices (n_s) where 4-length repeated substrings were found. n_s is thus the actual number of indices to output in order to give the result. These values are plotted against N in Figs. 2a, 2b, 2c and 2d.

Figs. 2a and 2c correspond to the case $m=10$. They clearly show that both t and n_s vary linearly with N, at least for high values of N. As mentioned in Proposition 4(a), n_s is bounded by $N \cdot g^k$. Note that the behavior is not linear for small values of N. This can be seen more clearly in Figs. 2b and 2d ($m=26$). During the evaluation of the boundaries, we have made the assumption that, in the worst case, all cliques appear on the string. In fact, for small N, only a small number of cliques did actually appear. This number depends upon N and the variation of n_s and t is no longer linear, although still linearly bounded (see Fig. 2b). For the same value of N, more cliques are obtained for $m=10$ than for $m=26$ and thus the asymptotic linear behaviour is observed sooner. One may also notice in Figs. 2b and 2d that the variation of t does not follow strictly that of n_s (actually for $m=26$, in that range of N, the variation of t is linear for $g=1$ and 1.5, and is less than N^2 for $g=2$ and $g=2.5$). This is shown in Fig. 3, where we plot the "relative time", defined as the ratio of time to the number of indices, versus N for $m=10$. We notice that, for a given g, the relative time decreases with N, as more and more cliques appear. Conversely, and for the same reason, at a given N, the relative time decreases with g. In many practical cases, R will be derived from a



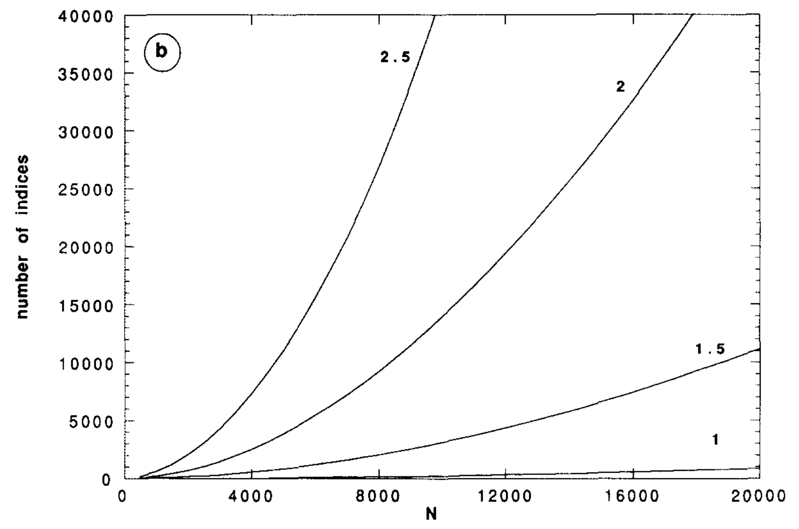


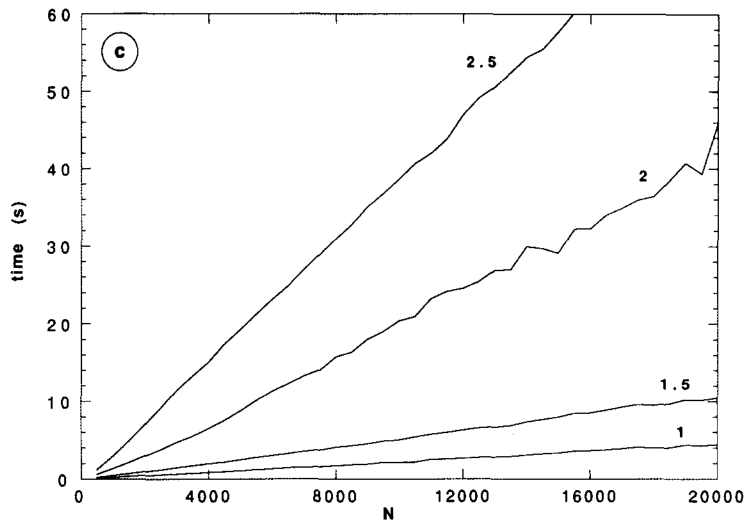
Fig. 2. (a) Averaged number n_s of repeated 4-length substrings plotted against the length N of random sequences. $m = 10$ and $\bar{g}$ varies from 1 to 2.5. (b) Averaged number n_s of repeated 4-length substrings plotted against the length N of random sequences. $m = 26$ and $\bar{g}$ varies from 1 to 2.5.

distance defined on the alphabet (e.g. $x \, R \, y$ iff distance$(x, y) \leqslant \alpha$, where α is a given threshold). In that case large values of the averaged degeneracy $\bar{g}$ are very unlikely.

3. Extensions to other problems

Two extensions may be considered.

The first one concerns repeated patterns in other structures such as arrays or trees. This has been considered by Karp et al. (1972) in the particular case where R is the identity. As an example, in order to find repeated $k \times k$ patterns in an $N \times N$ array, a variant of Lemma 1 was presented in which the right side was the conjunction of four



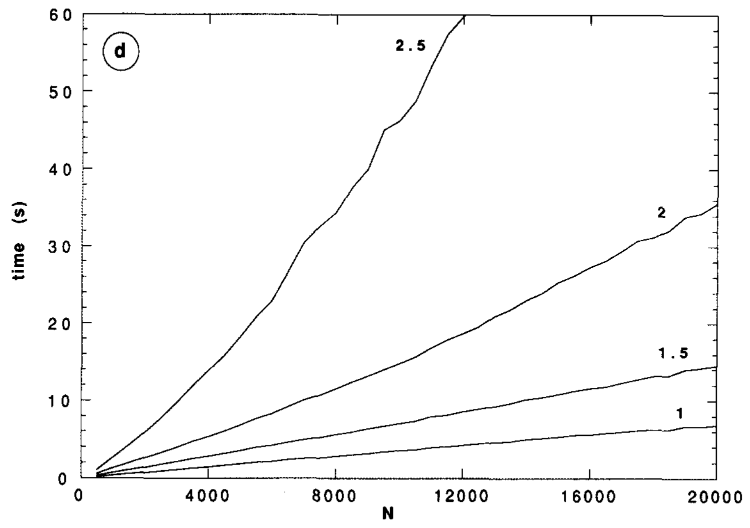


Fig. 2. (c) Averaged CPU time t plotted against the length N of random sequences. $m = 10$ and $\bar{g}$ varies from 1 to 2.5. (d) Averaged CPU time t plotted against the length N of random sequences. $m = 26$ and $\bar{g}$ varies from 1 to 2.5.

assertions instead of two. This leads, in our framework, to variants of Propositions 6, 7(a) and 8(a), in which four intersections have to be performed in order to obtain a set of cliques. As before, non-maximal cliques must then be removed. In this problem, each clique is represented by a set of index pairs.

The second extension concerns the use of more general relations between objects. Let us consider a set X of objects and suppose a recurrence relation expressed with 3 symmetric and reflexive relations R_1, R_2 and R_3:

$$x R_3 y \quad \Leftrightarrow \quad x R_1 y \text{ and } x R_2 y .$$

Then, one can show that the maximal cliques of the relation R_3 are obtained by computing all the intersections of pairs (C_1, C_2) of maximal cliques of R_1 and R_2 and by removing non-maximal cliques, as previously described. From this point of view, a variant of Lemma 1 is obtained by considering the relations R_1, R_2 and R_3 defined as follows:

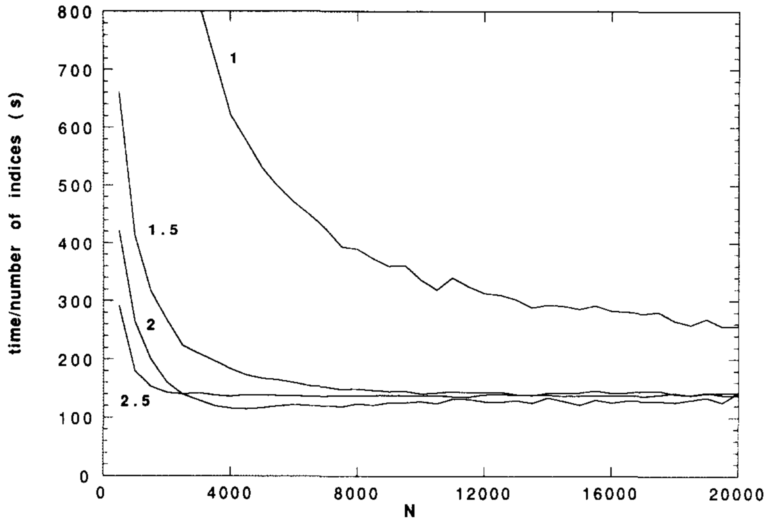


Fig. 3. Averaged t/n_s ratio plotted against the length N of random sequences. $k=4$, $m=26$ and $\bar{g}$ varies from 1 to 2.5.

$$R_1=R_a: \quad x\,R_a\,y \;\Leftrightarrow\; \forall i\in\{1,2,\dots,a\}, \quad V_i(x)\;R\;V_i(y)\,,$$

$$R_2=R_a^b: \quad x\,R_a^b\,y \;\Leftrightarrow\; \forall i\in\{1,2,\dots,a\}, \quad V_{i+b}(x)\;R\;V_{i+b}(y)\,,$$

$$R_3=R_{a+b}: \quad x\,R_{a+b}\,y \;\Leftrightarrow\; \forall i\in\{1,2,\dots,a+b\}, \quad V_i(x)\;R\;V_i(y)\,.$$

This variant allows us to search for maximal cliques of R_k in a dictionary of short strings, x standing for one string in the dictionary, rather than inside an N-length string.

4. Conclusion

In this paper we give a framework, some results, and some algorithms to address the problem of finding the maximal cliques of a relation R_k defined on labeled objects of size k, and derived from a symmetric and reflexive relation R on the alphabet. The proposed algorithms will find repeated k-length flexible patterns in a string and are extensions of the KMR algorithms. The significant parameter is the averaged degeneracy $\bar{g}$ of R, i.e., the averaged number of maximal cliques of R to which a symbol belongs. The theoretical worst case time complexity is linear with the size N of the string. It should be pointed out that these algorithms apply to the particular case of an alphabet containing s symbols plus a "don't care" symbol "#" which matches any other symbol (in this case $\bar{g}=2s/(s+1)$ if all symbols including "#" have an equal probability). We have already used successfully these algorithms in the field of molecular biology, where R is derived from amino-acid substitution matrices and the strings represent protein sequences. As previously published in (Landraud, 1989), it is relatively easy to add some constraints in the KMR algorithms in order, for instance, to look for patterns shared by several distinct sequences (e.g. Landraud, 1989) or to look for dyad symmetries in DNA sequences (e.g. Martinez, 1983). Of course, the same is true for the algorithms proposed here. We also hope that this framework will be useful in other pattern recognition problems such as image analysis, speech recognition and time series analysis.

References

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

Landraud, A., J.F. Avril and P. Chretienne (1989). An algorithm for finding a common structure shared by a family of strings. In: *IEEE Trans. Pattern Anal. Machine Intell.* 11, 890–895.

Martinez, H.M. (1983). An efficient method for finding repeats in molecular sequences. *Nucl. Acids Res.* 11, 4629–4634.