An algorithm that builds a set of strings given its overlap graph*

Marília D. V. Braga and João Meidanis

Institute of Computing, University of Campinas, P.O.Box 6176, 13084-971, Campinas, São Paulo, Brazil.
{mvbraga, meidanis}@ic.unicamp.br

Abstract. The k-overlap graph for a set of strings is a graph such that each vertex corresponds to a string and there is a directed edge between two vertices if there is an overlap of at least k characters between their corresponding strings. Given a directed graph G, an integer $k \geq 1$, and a finite alphabet Σ of at least two symbols, we propose an algorithm to obtain a set of strings C, written over Σ, such that G is its k-overlap graph. The algorithm runs in exponential time on the maximum degree of G, due to the size of the returned strings, but in polynomial time on k, $|\Sigma|$, and the size of the graph. A practical application of this algorithm is its use to prove the NP-hardness of *Minimum Contig Problems* family (*MCP*) and its variation *MCPr*, which are based on the DNA Fragment Assembly problem.

1 Introduction

Given an integer $k \geq 1$ and a set C of strings, written over an alphabet Σ, the k-overlap graph of C, denoted by $G_k(C)$, is the directed graph G such that $V(G) = C$ and $E(G) = \{uv \mid \text{there is an integer } \ell \geq k \text{ such that } last(u, \ell) = first(v, \ell)\}$, where we denote by $last(s, \ell)$ the last ℓ characters of s and by $first(s, \ell)$ the first ℓ characters of s [3].

We say that there is an overlap of ℓ characters between strings s and t when $last(s, \ell) = first(t, \ell)$. An example of overlap graph can be seen in Figure 1.

In 1992, Gusfield, Landau and Schieber [4] presented an efficient algorithm for the following problem. Given a set $C = \{s_1, s_2, s_3, \ldots, s_n\}$ of strings, find the greatest integer ℓ_{ij} such that $last(s_i, \ell_{ij}) = first(s_j, \ell_{ij})$, for all pair of strings $s_i, s_j \in C$, with $i \neq j$. Notice that this algorithm can be used to build $G_k(C)$ for any $k \geq 0$, because $s_i s_j \in E(G_k(C))$ if and only if $\ell_{ij} \geq k$.

Here we propose an algorithm that solves the inverse problem: given a directed graph G we build a set C' of strings over an alphabet Σ_G of size $|V(G)| + |E(G)|$ with the property that $G = G_1(C')$. For any $k \geq 1$ and for any finite alphabet Σ with at least two symbols, we also show how it is possible to modify C' and obtain a set of strings C, written over Σ, such that $G = G_k(C)$.

* Research supported by Brazilian agencies FAPESP (Grant 97/11629-2), and CNPq (Grant Pronex 664107/1997-4).

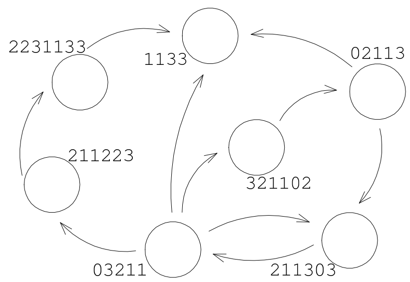


Fig. 1. 2-overlap graph for a set of strings written over the alphabet $\{0, 1, 2, 3\}$

A practical application of this algorithm is its use to prove the NP-hardness of *Minimum Contig Problems* family (MCP) and its variation $MCPr$ [3, 1], which are based on the DNA Fragment Assembly problem [6].

An ordered set of strings $S = (s_1, s_2, \ldots, s_{|S|})$ is called a k-path, for an integer $k \geq 0$, if each pair of consecutive strings s_{i-1}, s_i (where i is an integer between 2 and $|S|$) is such that there is an integer $\ell \geq k$ with $last(s_{i-1}, \ell) = first(s_i, \ell)$. We say that S is a k-path in a set of strings C if S is a subset of C.

Observe that if a graph G is a k-overlap graph for a set of strings C, a path in G corresponds to a k-path in C.

Given an integer k and a set of strings C written over an alphabet Σ, with $|\Sigma| = n$, the $MCP_n(k)$ is the problem of finding the smallest set of k-paths in C that *covers* all strings in C, that is, each string in C must be in exactly one k-path if it is not a proper substring of another string in C, or at most in one k-path if it is a proper substring of another string in C. Figure 2 shows a solution of an instance of MCP.

```
s TTACGCTA.........
t TTACGC...........
u ....GCTAGAGATC...
v ......TAGAGAT....
w ...........ATCGTT
  -----------------
  TTACGCTAGAGATCGTT
```

All input strings can be covered by one 2-path, (s, u, w), whose consensus is TTACGCTAGAGATCGTT.

Fig. 2. A solution of an instance of $MCP_4(2)$

We know that a DNA molecule is a double strand of nucleotides, identified by their bases `A`, `C`, `G`, `T`, organized in a way where an `A` in one strand is always linked to a `T` in the other, while a `C` in one strand is always linked to a `G` in the other. Moreover, the two strands in a DNA molecule have opposite orientations, that is, if the first strand is read from left to right, the second strand is read from right to left, and *vice-versa*. Because of this regular structure, one strand is called the **reverse complement** of the other strand in a molecule, meaning that it is easy to obtain the sequence of one strand given the other [6].

On the DNA Fragment Assembly approach [6], it is not possible to know which of the two strands originated each fragment, so it is necessary to consider the reverse complements of all fragment sequences to assemble the molecule.

Defining as $\overline{C}$ the set of all reverse complements of the strings in a set C, written over the alphabet $\{\texttt{A}, \texttt{C}, \texttt{G}, \texttt{T}\}$, the *MCPr* family of problems is a variation of *MCP* that finds the smallest set of k-paths in $C \cup \overline{C}$, where each string in C or its reverse complement must be in exactly one k-path if it is not a proper substring of another string in $C \cup \overline{C}$, or at most in one k-path if it is a proper substring of another string in $C \cup \overline{C}$. The solutions of an instance of *MCPr* can be seen in figure 3.

s	`TTACGCTA.........`	$\overline{s}$	`.........TAGCGTAA`
u	`....GCTAGAGATC...`	$\overline{u}$	`...GATCTCTAGC....`
w	`...........ATCGTT`	$\overline{w}$	`AACGAT...........`
	`-----------------`		`-----------------`
	`TTACGCTAGAGATCGTT`		`AACGATCTCTAGCGTAA`

All input strings can be covered by one 2-path, that can be the same (s, u, w) of figure 2 or $(\overline{w}, \overline{u}, \overline{s})$, whose consensus is `AACGATCTCTAGCGTAA`.

Fig. 3. Solutions of an instance of *MCPr* (2)

2 Directed edge colorings

We will now introduce a notion of coloring for directed graphs. We assume the basic definitions of standard graph theory [2].

Given a directed graph G, we define a *directed matching* as a set $M \subseteq E(G)$, such that for any two distinct edges $uv, xw \in M$, we have $u \neq x$ and $v \neq w$. Notice that the edges of M form a collection of disjoint directed paths and cycles in G. If a set $\mathcal{L} = \{M_1, M_2, \ldots, M_p\}$ of disjoint directed matchings covers all the edges of $E(G)$ we say that $\mathcal{L}$ is a *directed edge coloring* of G.

We denote by $\Delta(H)$ the maximum degree in an undirected graph H and by $\Delta^+(G)$ and $\Delta^-(G)$, respectively, the maximum indegree and maximum outdegree of a directed graph G. Observe that $p \geq \max\{\Delta^+(G), \Delta^-(G)\}$, where p is the cardinality of any directed edge coloring of G.

Theorem 1 *For every directed graph G there is a directed edge coloring $\mathcal{L} = \{M_1, M_2, \ldots, M_p\}$, such that $p = \max\{\Delta^+(G), \Delta^-(G)\}$.*

Proof:

Given a directed graph G we build $H = bip(G)$ as follows. For each vertex $v \in V(G)$ we include two vertices v', v'' in $V(H)$, so that $|V(H)| = 2|V(G)|$. For each edge $uv \in E(G)$, we include an edge $u'v''$ in $E(H)$, so that $|E(H)| = |E(G)|$. Notice that H is a bipartite undirected graph.

Given an undirected edge coloring $\mathcal{C} = \{M'_1, M'_2, \ldots, M'_p\}$ for $bip(G)$, consider $M_i = \{uv \mid u'v'' \in M'_i\}$. Since M'_i is a matching in $bip(G)$, we have that M_i is a directed matching in G. The matchings M'_i are disjoint and cover all the edges of $bip(G)$, and therefore the same is true for the directed matchings M_i with respect to G.

König's Theorem [2, p.103] states that the minimum number of colors (matchings) in an undirected coloring of a bipartite graph H is its maximum degree $\Delta(H)$. Since $\Delta(bip(G)) = \max\{\Delta^+(G), \Delta^-(G)\}$, there exists a directed edge coloring for G with $\max\{\Delta^+(G), \Delta^-(G)\}$ colors.

□

Figure 4 shows an example of $bip(G)$ and a directed edge coloring for a directed graph G.

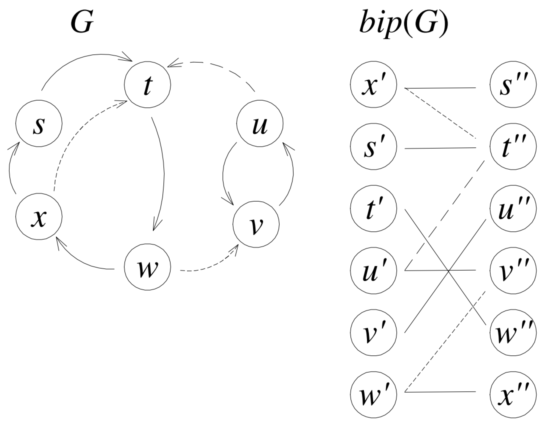


Fig. 4. A directed graph G, its bipartite version $bip(G)$, and a directed edge coloring

3 The algorithm

Given a directed graph G with $|V(G)| = n$ and $|E(G)| = m$, and an integer $k \geq 1$, we propose an algorithm that runs in time polynomial in n and m and exponential in $p = \max\{\Delta^+(G), \Delta^-(G)\}$ and constructs a set C of strings, written over any alphabet with at least two symbols, such that $G = G_k(C)$ and $|s|$ is $O(k.\log_2(m+n).2^p)$ for every $s \in C$.

The algorithm is composed of two parts:

1. First we build a set C' of strings, written over alphabet $\Sigma_G = \{0, 1, 2, ..., m+n-1\}$, such that $G = G_1(C')$ and $|s'| \leq 2^{p+1} - 1$ for every $s' \in C'$. For each $v \in V(G)$, the string generated is of the form $\mathit{pref}(v).B(v).\mathit{suf}(v)$, where $B(v)$ is a unique character, not occuring in $\mathit{pref}(v)$ or in $\mathit{suf}(v)$ or in any other string of C'. This implies that any overlap between strings corresponding to two vertices u and v is in fact an overlap between $\mathit{suf}(u)$ and $\mathit{pref}(v)$. This also guarantees that no string in C' is a substring of another string in C'.
2. Then we use a function f to code the strings in C', forming a set C of strings written over an arbitrary alphabet with at least two symbols, satisfying $G = G_k(C)$, because the properties of C' with respect to G are maintained in C, that is, for each $v \in V(G)$, the new string is of the form $f(\mathit{pref}(v)).f(B(v)).f(\mathit{suf}(v))$, where $f(B(v))$ is a string, not occuring in $f(\mathit{pref}(v))$ or in $f(\mathit{suf}(v))$ or in any other string of C. This implies that any overlap between strings corresponding to two vertices u and v is in fact an overlap between $f(\mathit{suf}(u))$ and $f(\mathit{pref}(v))$. This also guarantees that no string in C is a substring of another string in C.

3.1 Part 1

Algorithm 1 *Construction of C', given directed graph G:*

1. *For each $v \in V(G)$ do:*
   - 1.1. $\mathit{pref}(v) \leftarrow \varepsilon$;
   - 1.2. $\mathit{suf}(v) \leftarrow \varepsilon$;
2. *Find a minimum directed edge coloring $\mathcal{L} = \{M_1, M_2, \ldots, M_p\}$ for G, where $p = \max\{\Delta^+(G), \Delta^-(G)\}$;*
3. $I \leftarrow 0$;
4. *For each $M_i \in \mathcal{L}$ do:*
   - 4.1. *For each edge $uv \in M_i$ do:*
     - 4.1.1. $\mathit{pref}(v) \leftarrow \mathit{pref}(v).I.\mathit{suf}(u)$;
     - 4.1.2. $\mathit{suf}(u) \leftarrow \mathit{pref}(v)$;
     - 4.1.3. $I \leftarrow I + 1$;
5. $C' \leftarrow \emptyset$;
6. *For each $v \in V(G)$ do:*
   - 6.1. $C' \leftarrow C' \cup \{\mathit{pref}(v).I.\mathit{suf}(v)\}$;
   - 6.2. $I \leftarrow I + 1$;
7. *Return C';*

Figure 5 shows an example of execution of the algorithm.

Denote by $A(u,v)$ the value of I when edge uv is being processed in step 4.1.1. For every $uv \in E(G)$, we have $0 \leq A(u,v) \leq m-1$ and $A(u,v) \neq A(w,x)$ if $uv \neq wx$.

We will prove the correctness of the algorithm and then analyze the sizes of the strings generated. To prove the correctness, we need to show that:

(i) If $uv \in E(G)$, then there is an integer $\ell \geq 1$ such that $last(suf(u),\ell) = first(pref(v),\ell)$.

Given edge $uv \in E(G)$, at the moment it was processed in step 4.1.2 we created an overlap between $suf(u)$ and $pref(v)$ consisting of at least one character: $A(u,v)$. After that point in the execution of the algorithm, $suf(u)$ grows only to the left, while $pref(v)$ grows only to the right, maintaining the overlap throughout the end of the algorithm. Therefore, there is an integer $\ell \geq 1$ with $last(suf(u),\ell) = last(pref(v),\ell)$.

(ii) If $last(suf(u),\ell) = first(pref(v),\ell)$ for some integer $\ell \geq 1$ and two vertices $u, v \in V(G)$, then $uv \in E(G)$.

Let i be the outdegree of u, and let uv_1, uv_2, ..., uv_i be the edges going out of u, in the order they are processed by the algorithm, so that $A(u,v_i) > \ldots > A(u,v_2) > A(u,v_1)$. Character $A(u,v_i)$ appears only once in $suf(u)$ and is the largest (viewed as an integer) among all characters in $suf(u)$. In addition, for each i' such that $1 \leq i' < i$, character $A(u,v_{i'})$ appears only once to the right of $A(u,v_{i'+1})$ and is the largest integer in that portion of $suf(u)$. It follows that, for any suffix s of $suf(u)$, the largest integer in s is a character of the form $A(u,v_{i'})$ with $1 \leq i' \leq i$.

Analogously, if j is the indegree of v and u_1v, u_2v, ..., u_jv are the edges going into v in the order they are processed by the algorithm, we have that the largest integer in any prefix of $pref(v)$ is a character of the form $A(u_{j'},v)$ with $1 \leq j' \leq j$.

If $last(suf(u),\ell) = first(pref(v),\ell)$ then the largest character in this string is $A(u,v_{i'})$, for some i' between 1 and i, and also $A(u_{j'},v)$, for some j' between 1 and j. It follows that $A(u,v_{i'}) = A(u_{j'},v)$, so $u = u_{j'}$ and $v = v_{i'}$, that is, edge uv exists in G.

This completes the correctness proof.

The next step is to prove that $|s| \leq 2^{p+1}-1$ for every $s \in C'$, where $p = \max\{\Delta^+(G), \Delta^-(G)\}$. Recall that $\mathcal{L} = \{M_1, M_2, \ldots, M_p\}$ is a directed coloring of G.

Denote by $size(i)$ the length of the longest string obtained after processing all the edges of M_i. We have $size(0) = 0$ because $pref(.)$ and $suf(.)$ are empty in the beginning.

For every vertex $v \in V(G)$, it is clear that each of the strings $pref(v)$ and $suf(v)$ is modified at most once during the processing of each M_i, because of the way directed matchings were defined. The modification consists in concatenating a suffix with a single character and with a prefix, hence we have $size(i) \leq$

$2.size(i-1)+1$. It is easy to prove by induction that $size(i) \leq 2^i - 1$, implying that after all edges are processed we have $size(p) \leq 2^p - 1$.

The representative of a vertex v has length $|pref(v)|+|suf(v)|+1 \leq 2.size(p)+1 \leq 2^{p+1}-1$, completing this analysis.

Computational complexity Let us compute the complexity of the algorithm for an instance G with $|V(G)| = n$, $|E(G)| = m$ and $p = \max\{\Delta^+(G), \Delta^- G\}$.

In step 2 we find a directed edge coloring of G. As we have seen in the proof of Theorem 1, this can be done by solving an undirected bipartite edge coloring problem, which has complexity $O(pm)$ [5].

The other time-stressing steps are step 4, which requires $\sum_{i=1}^{|\mathcal{L}|} |M_i|.(2^{p+1} - 1) = O(2^p m)$ and step 6, which takes $O(2^p n)$ time. Therefore, the total running time is $O(2^p(m+n))$.

Notice that the exponential component of the running time is due solely to the processing of exponentially large strings. If the strings could be implicitly represented as a concatenation of pointers to other strings, the output and the algorithm would be polynomial.

3.2 Part 2

In the second part of the algorithm, we use a function f, with special properties, to code the strings of C', forming a set C of strings written over an arbitrary, given alphabet with at least two symbols, such that $G = G_k(C)$ and $|s|$ is $O(k.\log_2(m+n).2^p)$ for each $s \in C$, where $m = |E(G)|$ and $n = |V(G)|$.

Before constructing function f, we introduce the concepts of homomorphism and occurrence.

Given alphabets Σ and Υ, a **homomorphism** $g : \Sigma^* \longrightarrow \Upsilon^*$ is a function such that, for every $u, v \in \Sigma^*$, we have $g(uv) = g(u)g(v)$. As a consequence of this property, we have $g(\varepsilon) = \varepsilon$.

Given two strings s and t, we say that there is an occurrence of t in s when either t is a substring of s or there is an integer $\ell \geq 1$ such that $last(s, \ell) = first(t, \ell)$.

Therefore, we say that a homomorphism $g : \Sigma^* \longrightarrow \Upsilon^*$ **preserves occurrences** when g is a one-to-one function such that, for all $c \in \Sigma$, $g(c)$ has a fixed size, denoted by $|g|$ and, given $s, t \in \Sigma^*$, we have:

- Given any integer $\ell \geq 0$, we have $last(s, \ell) = first(t, \ell)$ if, and only if, $last(g(s), \ell|g|) = first(g(t), \ell|g|)$;
- t is a proper substring of s if, and only if, $g(t)$ is a proper substring of $g(s)$.

Algorithm 2 *Codification of the strings in C' into C, written over a given alphabet Σ, with $|\Sigma_G| \geq |\Sigma| \geq 2$, based on the given integer $k \geq 1$.*

1. *Take a homomorphism $g : \Sigma_G^* \longrightarrow \Sigma^*$, such that g preserves occurrences and $|g| \leq 2\lceil \log_{|\Sigma|}(m+n) \rceil + 1$;*
2. *Calculate the integer $k' = \lceil k/|g| \rceil$;*

3. *Take the homomorphism* $h_{k'} : \Sigma_G^* \longrightarrow \Sigma_G^*$*, such that* $h_{k'}(c) = c^{k'}$ *for each* $c \in \Sigma_G$*;*
4. *Construct the function* $f : \Sigma_G^* \longrightarrow \Sigma^*$*, defined as* $f(s) = g(h_{k'}(s))$ *for each* $s \in \Sigma_G^*$*;*
5. *Return the set* $C = \{f(s) \mid s \in C'\}$*;*

The correctness of the algorithm is a consequence of the following theorems. To make the proofs easier, we created a special notation for the alphabets we will use. Given an integer $m' \geq 1$, we denote by $\Sigma_{m'}$ the alphabet $\{0, 1, 2, \ldots, m'-1\}$, so that $|\Sigma_{m'}| = m'$.

Furthermore, given two integers m' and n' with $2 \leq n \leq m'$, we define a one-to-one function $b_{m'n'}$ such that, for all $c \in \Sigma_{m'}$, $b_{m'n'}(c)$ is the base n' representation of c with exactly $\lceil \log_{n'} m' \rceil$ characters (leading zeroes are used where necessary).

Theorem 2 *Given a homomorphism* $g : \Sigma^* \longrightarrow \Sigma^*$*, such that, for all* $c \in \Sigma$*,* $g(c)$ *has fixed size* $|g|$ *and is defined as* $g(c) = g'(c).y$*, where* y *is a constant string and* $g'(c)$ *has also fixed size, and if in addition there is no occurrence of* $g'(c)$ *in* y*, and no occurrence of* y *in* $g'(c)$*, then* g *preserves occurrences.*

Proof:

From the definition of g we deduce that, given $s \in \Sigma^*$, if we take a string x such that $g(s) = uxv$ and, in addition, x begins with $g'(c)$ (for some $c \in \Sigma$) and ends in y, then $|x|$ is a multiple of $|g|$ and $x \in \mathrm{Im}(g)$.

To show that g preserves occurrences, given two strings $s, t \in \Sigma^*$, we need to show that:

- Given any integer $\ell > 0$, we have $last(s, \ell) = first(t, \ell)$ if, and only if, $last(g(s), \ell|g|) = first(g(t), \ell|g|)$.

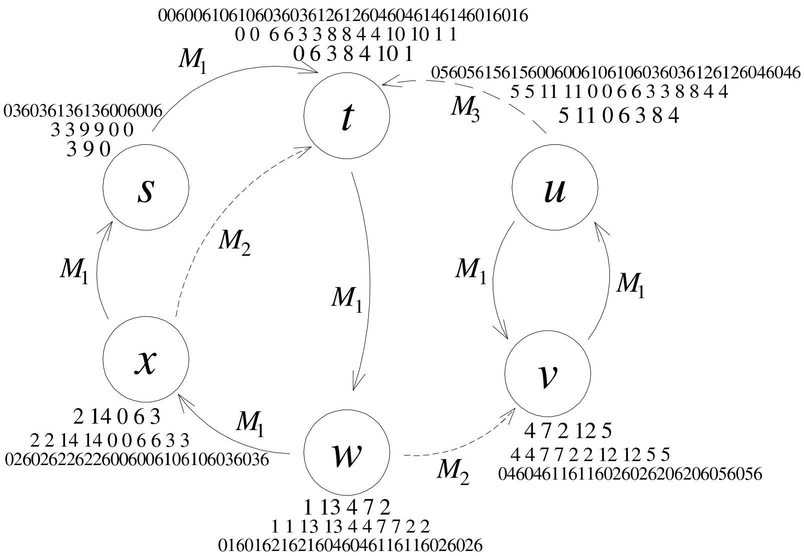
Suppose that $s = uv$ and $t = vw$, with $|v| = \ell$. We know that $g(s) = g(u)g(v)$ and $g(t) = g(v)g(w)$. Therefore, there exists the overlap $g(v)$ between $g(s)$ and $g(t)$. Thus, we have $|g(v)| = |v||g| = \ell|g|$.

On the other hand, suppose that $g(s) = uv$ and $g(t) = vw$. We can be certain that v begins with $g'(t_1)$, because $g(t)$ begins with $g'(t_1)$, and ends in y, because $g(s)$ ends in y. We conclude that $|v|$ is a multiple of $|g|$ and $v \in \mathrm{Im}(g)$. Furhtermore, we have $u, w \in \mathrm{Im}(g)$, with $s = g^{-1}(u)g^{-1}(v)$ and $t = g^{-1}(v)g^{-1}(w)$. Therefore, if $|v|/|g| = \ell$, we have $last(g(s), \ell|g|) = first(g(t), \ell|g|)$ and, in consequence, $last(s, \ell) = first(t, \ell)$.

- t is a proper subtring of s if, and only if, $g(t)$ is a proper substring of $g(s)$.

Now suppose that $s = utv$. We know that $g(s) = g(u)g(t)g(v)$ and, therefore, we can say that $g(t)$ is a substring of $g(s)$.

Analogously, suppose that $g(s) = u.g(t).v$. Certainly v ends in y, because $g(s)$ ends in y, and v begins with $g'(s_j)$, for some integer j, because $g(t)$ ends



$|V(G)| = 6$, $|E(G)| = 9$ (then $\Sigma_G = \{0, 1, 2, \ldots 14\}$) and $\max\{\Delta^+(G), \Delta^-(G)\} = 3$

$\mathcal{L} = \{M_1, M_2, M_3\}$ with $M_1 = \{st, tw, wx, xs, uv, vu\}$, $M_2 = \{xt, wv\}$, and $M_3 = \{ut\}$

$A(s,t) = 0$	$A(x,s) = 3$	$A(x,t) = 6$	$B(s) = 9$	$B(v) = 12$
$A(t,w) = 1$	$A(u,v) = 4$	$A(w,v) = 7$	$B(t) = 10$	$B(w) = 13$
$A(w,x) = 2$	$A(v,u) = 5$	$A(u,t) = 8$	$B(u) = 11$	$B(x) = 14$

Iterations on step 4 of algorithm 1:

Processing M_1:	Processing M_2:
(i) $\mathit{suf}(s) = \mathit{pref}(t) = 0$	(vii) $\mathit{suf}(x) = \mathit{pref}(t) = 0\ 6\ 3$
(ii) $\mathit{suf}(t) = \mathit{pref}(w) = 1$	(viii) $\mathit{suf}(w) = \mathit{pref}(v) = 4\ 7\ 2$
(iii) $\mathit{suf}(w) = \mathit{pref}(x) = 2$	
(iv) $\mathit{suf}(x) = \mathit{pref}(s) = 3$	
(v) $\mathit{suf}(u) = \mathit{pref}(v) = 4$	Processing M_3:
(vi) $\mathit{suf}(v) = \mathit{pref}(u) = 5$	(ix) $\mathit{suf}(u) = \mathit{pref}(t) = 0\ 6\ 3\ 8\ 4$

Defining g on step 1 of algorithm 2, for $|\Sigma| = 7$ and $k = 4$:

$g(0) = 006$	$g(3) = 036$	$g(6) = 106$	$g(9) = 136$	$g(12) = 206$
$g(1) = 016$	$g(4) = 046$	$g(7) = 116$	$g(10) = 146$	$g(13) = 216$
$g(2) = 026$	$g(5) = 056$	$g(8) = 126$	$g(11) = 156$	$g(14) = 226$

Calculating the integer k' on step 2 of algorithm 2 as $k' = \lceil k/|g| \rceil = \lceil 4/3 \rceil = 2$, the homomorphism $h_{k'}$ on step 3 of algorithm 2 is $h_2(c) = c^2$ for each $c \in \Sigma_G$.

Fig. 5. Executing the algorithm for $|\Sigma| = 7$ and $k = 4$

in y. In addition, u also ends in y, because $g(t)$ begins with $g'(t_1)$ and u begins with $g'(s_1)$, because $g(s)$ begins with $g'(s_1)$. With this, we show that $|u|$ e $|v|$ are multiples of $|g|$ and we have $u, v \in \mathrm{Im}(g)$. Therefore, we conclude that $s = g^{-1}(u)g^{-1}(g(t))g^{-1}(v) = g^{-1}(u).t.g^{-1}(v)$, which implies that t is a substring of s.

□

Theorem 3 *For every pair of integers m', n' such that $m' \geq n' \geq 2$, there is a homomorphism $g : \Sigma_{m'}^* \longrightarrow \Sigma_{n'}^*$ preserving occurrences.*

Proof:

Take integers m', n', with $m' \geq n'$, and consider two cases: $n' \geq 3$, and $n' = 2$.

1. In the first case, for any pair of integers m', n', with $m' \geq n' \geq 3$, we take $y = n' - 1$. For all $c \in \Sigma_{m'}$, define $g(c) = b_{m'(n'-1)}(c).y$, so that $|g| = \lceil \log_{n'-1} m' \rceil + 1$. Since $b_{m'(n'-1)}$ is one-to-one, g is also one-to-one, because y is a constant string.
   Note that, for all $c \in \Sigma_{m'}$, the string $b_{m'(n'-1)}(c)$ has fixed size and y is a constant string, with no occurrence of $b_{m'(n'-1)}(c)$ in y, and no occurence of y in $b_{m'(n'-1)}(c)$.
   Therefore, by Theorem 2, the homomorphism g preserves occurrences.
2. In the second case, for any $m' \geq 2$, define $g(c) = 1.b_{m'2}(c).1.0^{i+1}$, for all $c \in \Sigma_{m'}$, where $i = \lceil \log_2 m' \rceil$, resulting in $|g| = 2i + 3$. Since $b_{m'2}$ is one-to-one, so is g, because the strings 1 and 1.0^{i+1} are constant.
   Note that, for all $c \in \Sigma_{m'}$, the string $1.b_{m'2}(c).1$ has fixed size and 0^{i+1} is a constant string, with no occurrence of $1.b_{m'2}(c).1$ in 0^{i+1}, and no occurrence of 0^{i+1} in $1.b_{m'2}(c).1$.
   Therefore, also by Theorem 2, the homomorphism g preserves occurrences.

In both cases we have $|g| \leq 2\lceil \log_{n'} m' \rceil + 1$.

□

Theorem 4 *Given any alphabet Σ, the homomorphism $h_k : \Sigma^* \longrightarrow \Sigma^*$ such that $h_k(c) = c^k$ for all $c \in \Sigma$, preserves occurrences, for all $k \geq 1$.*

Proof:

Taking $k \geq 1$, we know that h_k is one-to-one, with $|h_k| = k$.

Consider strings $s, t \in \Sigma^*$ with $s = c_1 c_2 \ldots c_{|s|}$ and $t = d_1 d_2 \ldots d_{|t|}$. Then, we have $h_k(s) = h_k(c_1)h_k(c_2)\ldots h_k(c_{|s|})$ and $h_k(t) = h_k(d_1)h_k(d_2)\ldots h_k(d_{|t|})$.

To show that h_k preserves occurrences, we need to show that:

- Given an integer $\ell > 0$, we have $last(s, \ell) = first(t, \ell)$ if, and only if, $last(h_k(s), \ell k) = first(h_k(t), \ell k)$.

Suppose that $s = uv$ and $t = vw$, with $|v| = \ell$. Then $h_k(s) = h_k(u)h_k(v)$ and $h_k(t) = h_k(v)h_k(w)$. It follows that there is overlap $h_k(v)$ between $h_k(s)$ e $h_k(t)$. Therefore, we have $last(h_k(s), \ell k) = first(h_k(t), \ell k)$.

On the other hand, suppose $h_k(s) = uv$ and $h_k(t) = vw$. If $u, v, w \notin \mathrm{Im}(h_k)$, we would have $u = c_1^k c_2^k \ldots c_{i-1}^k c_i^j$, $v = c_i^{k-j} c_{i+1}^k c_{i+2}^k \ldots c_{|s|}^k = d_1^k d_2^k \ldots d_{\ell-1}^k d_\ell^m$ and $w = d_\ell^{k-m} d_{\ell+1}^k d_{\ell+2}^k \ldots d_{|t|}^k$, where $1 \le j < k$ and $1 \le i < |s|$, and, analogously, $1 \le m < k$ and $1 \le \ell < |t|$.

Then, we would have necessarily $\ell - 1 = |s| - (i+1) + 1 = |s| - i$ and $m = k - (j+1) + 1 = k - j$, implying that $v = c_i^m c_{i+1}^k c_{i+2}^k \ldots c_{|s|}^k = d_1^k d_2^k \ldots d_{|s|-i}^k d_{|s|-i+1}^m$ and, therefore, we would have $c_i = d_1 = c_{i+1} = d_2 = c_{i+2} = \ldots = d_{|s|-i} = c_{|s|} = d_{|s|-i+1}$. But then we could take strings $u' = c_1^k c_2^k \ldots c_{i-1}^k$, $v' = c_i^k c_{i+1}^k c_{i+2}^k \ldots c_{|s|}^k = d_1^k d_2^k \ldots d_{\ell-1}^k d_\ell^k$ and $w' = d_{\ell+1}^k d_{\ell+2}^k \ldots d_{|t|}^k$, which satisfy $u', v', w' \in \mathrm{Im}(h_k)$. In conclusion, if $|v'|/k = \ell$, we have $last(h_k(s), \ell k) = first(h_k(t), \ell k)$ and hence $last(s, \ell) = first(t, \ell)$.

– t is a proper substring of s if, and only if, $h_k(t)$ is a proper substring of $h_k(s)$.

Now suppose that $s = utv$. We know that $h_k(s) = h_k(u)h_k(t)h_k(v)$ and, therefore, $h_k(t)$ is a proper substring of $h_k(s)$.

Conversely, suppose that $h_k(s) = u.h_k(t).v$. If $u, v \notin \mathrm{Im}(h_k)$, we would have $u = c_1^k c_2^k \ldots c_i^k c_{i+1}^j$ and $v = c_{\ell-1}^{k-j} c_\ell^k c_{\ell+1}^k \ldots c_{|s|}^k$, where $1 \le j < k$ and $i < \ell$, with $i + |t| + \ell + 1 = |s|$.

Then, $h_k(s) = c_1^k c_2^k \ldots c_i^k c_{i+1}^j . d_1^k d_2^k \ldots d_{|t|}^k c_{\ell-1}^{k-j} c_\ell^k c_{\ell+1}^k \ldots c_{|s|}^k$ and, therefore, we would have $c_{i+1} = d_1 = d_2 = \ldots = d_{|t|} = c_{\ell-1} = c$. But then we could take strings $u' = c_1^k c_2^k \ldots c_i^k$, $x = c^k$ and $v' = c_\ell^k c_{\ell+1}^k \ldots c_{|s|}^k$, satisfying $h_k(s) = u' h_k(t) x v' = u' x h_k(t) v'$. It follows that is is always possible to choose u, v in $\mathrm{Im}(h_k)$ with $s = h_k^{-1}(u).t.h_k^{-1}(v)$. In other words, t is a substring of s.

□

Given a set of strings C, we denote by $T(C)$ the set formed by every string of C that is not a proper substrings of other string in C. The following result is important in the proof of NP-hardness of the family MCP [3, 1].

Theorem 5 *Given integers $m' \ge n' \ge 2$, a homomorphism $g : \Sigma_{m'}^* \longrightarrow \Sigma_{n'}^*$ that preserves occurrences, and a set of strings C', written over $\Sigma_{m'}$, the set $C = g(C')$, satisfies:*

1. *$T(C) = g(T(C'))$ and*
2. *for all pairs of integers k and k' with $(k'-1)|g| < k \le k'|g|$, we have $G_k(C) = G_{k'}(C')$.*

Proof:

Since $C = g(C')$, we can restrict the homomorphism g to the bijection $g : C' \longrightarrow C$.

(1) Since g preserves occurrences, g preserves in C the same substring relationships that existed in C'. Therefore, $T(g(C')) = g(T(C'))$, that is, g commutes with T. It follows that $T(C) = g(T(C'))$.

(2) Take $u, v \in C'$ and consider $g(u), g(v) \in C$. By definition we know that $(u, v) \in E(G_{k'}(C'))$ implies $last(u, \ell) = first(v, \ell)$ and $\ell \ge k'$. Since g preserves

occurrences, we also have $last(g(u), \ell|g|) = first(g(v), \ell|g|)$ and $\ell|g| \geq k'|g|$, that is, for all integers $k \leq k'|g|$, we have $(g(u), g(v)) \in E(G_k(C))$. On the other hand, if $(u, v) \notin E(G_{k'}(C'))$, and there exists ℓ such that $last(u, \ell) = first(v, \ell)$, then $\ell \leq k'-1$. Consequently, if there is ℓ' such that $last(g(u), \ell'|g|) = first(g(v), \ell'|g|)$, then $\ell' \leq k'-1$, because otherwise we would have $last(u, \ell') = first(v, \ell')$, and the edge (u, v) would be present in $G_{k'}(C')$, which is a contradiction. Therefore, for all integers $k > (k'-1)|g|$, we have $(g(u), g(v)) \notin E(G_k(C))$.

We conclude that, for all pairs of integers k e k' such that $(k'-1)|g| < k \leq k'|g|$, we have $G_k(C) = G_{k'}(C')$.

□

This guarantees the correctness of the algorithm 2, once we proved that functions g (on step 1) and $h_{k'}$ (on step 3) preserve occurrences and, by Theorem 5, we can say that $G = G_1(C') = G_{k'}(h_{k'}(C')) = G_k(g(h_{k'}(C'))) = G_k(C)$.

Computational complexity Notice that each string had its size multiplied by $|g||h_{k'}|$, where $|g| \leq 2\lceil \log_{|\Sigma|}(n+m) \rceil + 1$ and $|h_{k'}| = \lceil h/|g| \rceil \leq k/|g| + 1$. Then $|h_{k'}||g| \leq k + |g| \leq k + 2\lceil \log_{|\Sigma|}(n+m) \rceil + 1 \leq k + 2\lceil (\log_2(n+m))/(\log_2|\Sigma|) \rceil + 1$.

The total running time is given by the size of the string set multiplied by the size of each string, that is $O(n2^p(k + (\log_2(n+m))/(\log_2|\Sigma|)))$.

4 Final Remarks

Although the algorithm runs in exponential time on the maximum degree of the input graph, it can be useful on studying problems that involve strings and graphs.

Considering specifically the *MCP* family of problems [3], we used this algorithm to reduce the Hamiltonian path problem, restricted to graphs with maximum degree smaller than or equal to 3, to the decision version of each problem in *MCP*, proving that *MCP* is NP-hard [1].

In addition, we applied occurrence-preserving homomorphisms to reduce the problems in *MCP* to the problems in *MCPr* [3], showing that this family is also NP-hard [1].

References

1. M. D. V. Braga. Grafos de Seqüências de DNA. Master's thesis, Instituto de Computação, UNICAMP, Campinas, SP, November 2000. In Portuguese at http://www.ic.unicamp.br/~meidanis/research/fa.html.
2. R. Diestel. *Graph Theory*. Springer-Verlag New York, Inc, 1997.
3. C. E. Ferreira, C. C. de Souza, and Y. Wakabayashi. Rearrangement of DNA fragments: a branch-and-cut algorithm. *Discrete Applied Mathematics*, 2001. To appear.
4. D. Gusfield, G. M. Landau, and B. Schieber. An efficient algorithm for the all pairs suffix-prefix problem. *Information Processing Letters*, 41(4):181–185, March 1992.

5. A. Schrijver. Bipartite edge coloring in $O(\Delta m)$ time. *SIAM Journal on Computing*, 28(3):841–846, 1998.
6. J. C. Setubal and J. Meidanis. *Introduction to Computational Molecular Biology*. PWS Publishing Company, 1997.