Validating and Decomposing Partially Occluded Two-Dimensional Images (Extended Abstract)[1]

Costas S. Iliopoulos[1,2] and James F. Reid[1,3]

[1] Algorithm Design Group, Department of Computer Science
King's College London, London WC2R 2LS, UK.
[2] School of Computing, Curtin University of Technology
Perth, WA 6102, Australia.
[3] Dipartimento di Elettronica e Informatica
Università degli Studi di Padova
Via Gradenigo 6/a, 35131 Padova, Italy.

e-mail: {csi,jfr}@dcs.kcl.ac.uk

Abstract. A partially occluded scene in an image consists of a number of objects that are partially obstructed by others. Validating a partially occluded image consists of generating a sequence of concatenated and possibly overlapping objects that corresponds to the input image. The algorithm presented here validates a two-dimensional image X of size $r \times s$ over a set of k objects of identical size $m \times m$ in $O(mrs)$ time.

Key words: String algorithms, image processing, occlusion analysis, pattern recognition.

1 Introduction

The study of repetitive structures in sequences (strings) plays a key role in information processing and more generally in computer science. This has lead to a generalization of notions concerning repetitions in sequences. The periodicity of a string was the key element in the design of the famous pattern matching algorithm by Knuth, Morris and Pratt, [KMP-77]. A related notion is the one of a cover of a string. A substring w of a string x is called a *cover* of x if x can be constructed by concatenations and superpositions of w. As a result, many sequential and parallel algorithms have been developed concerning the covering of a string. Among the sequential algorithms, Apostolico, Farach and Iliopoulos [AFI-91] solved the problem of computing the shortest cover of a given string, similarly Moore and Smyth [MS-95] solved the problem of computing all the covers of a given string both in linear time. These

[1]C.S. Iliopoulos was partially supported by EPSRC grants GR/F 00898, GR/L 19362 and GR/J 17844, NATO grant CRG 900293, and MRC grant G9115730. J.F. Reid was supported by a Marie Curie fellowship of the European Commission Training and Mobility of Researchers (TMR) Programme.

efficient solutions for string covering problems have applications to DNA sequencing by hybridization, see [DS-96] and [PL-94].

This paper focuses on an application of the string covering techniques to image processing and the analysis of images composed of know objects obstructing each other. Decomposing partially occluded images is a classical problem in computer vision and its computational complexity is exponential. There are many artificial intelligence and neural network solutions to this problem, see for example [BC-94]. Here we present a theoretical study on the analysis of images composed from a given set of objects, where some of the appearing objects may be partially occluded by other ones. Thus we restrict our attention on the occlusion problem by focusing only on discrete images and convex objects, and their efficient solutions are based on the study of the repetitive structures of the input. The results and solutions presented here provide the foundations for practical solutions to this problem. This problem was first approached by only considering one-dimensional images (strings). A linear sequential on-line algorithm was produced by Iliopoulos and Simpson [IS-97] and an optimal parallel version was also produced, see [IR-97].

In the following, we will consider the family of two dimensional images (considered as two-dimensional arrays of strings), that we call *valid images*; given a set of objects $\{S_1, \ldots, S_k\}$ and a special "background" symbol denoted #, an image X of size $r \times s$ is a valid image, if X is iteratively obtained from an initial string Z of size $r \times s$ consisting only of #'s by substituting substrings of Z by some objects S_i, for some i. We will be focusing in designing algorithms for testing two-dimensional images for validity, under restricted sets of objects, i.e. square objects of the same size.

Here we present an algorithm for validating a two-dimensional image X of size $r \times s$ over a fixed number k of objects S_i of equal size $m \times m$ in $O(mrs)$ time.

The paper is organised as follows. In the next section we present the basic definitions for strings and partially occluded images. In Section 3 we describe the data structures and the main techniques used in the algorithm and finally in Section 4 we present our conclusions and open problems.

2 Preliminaries

2.1 String definitions in one and two dimensions

A *string* (or word) is a sequence of zero or more symbols drawn from an alphabet Σ, which consists of a finite number of symbols. The set of all strings over Σ is denoted by Σ^*. The string of length zero is the *empty string* ϵ and a string x of length $n > 0$ is represented by $x_0 x_1 \cdots x_{n-1}$, where $x_i \in \Sigma$ for $0 \leq i \leq n-1$. A string w is said to be *substring* of x if and only if $x = uwv$ for some $u, v \in \Sigma^*$. A string w is a *prefix* of x if and only if $x = wu$ for some $u \in \Sigma^*$; if u is not empty then w is a called a *proper prefix* of x. Similarly, w is a *suffix* of x if and only if $x = uw$ for some $u \in \Sigma^*$; if u is not empty then w is called a *proper suffix* of x. Additionally $prefix_k(x)$ denotes the first k symbols of x and $suffix_k(x)$ denotes the last k symbols of x. The string xy is a *concatenation* of two strings x and y. The concatenation of k copies of x is denoted by x^k. For two strings $x = x_0 \cdots x_{n-1}$ and $y = y_0 \cdots y_{m-1}$ such that $x_{n-i} \cdots x_{n-1} = y_0 \cdots y_i$ for some $i \geq 1$ (that is, such that x has a suffix equal to a prefix of y), the string $x_0 \cdots x_{n-1} y_i \cdots y_{m-1}$ is said to be a *superposition* of x and y.

Alternatively, we may say that x *overlaps* with y. A substring w of x is called a *cover* of x if x can be constructed by concatenations and superpositions of w.

A *two-dimensional string* is an $r \times s$ array of symbols drawn from Σ. We will refer to a two-dimensional string as a *two-dimensional array* or a *two-dimensional image* in the sequel. We represent an $r \times s$ array X by $X[0..r-1, 0..s-1]$. A two-dimensional $p \times q$ array Y is said to be a *sub-array* or a *sub-image* of X if the upper left corner of Y can be aligned with $X[i,j]$, i.e. $Y[0..p-1, 0..q-1] = X[i..i+p-1, j..j+q-1]$, for some $0 \leq i \leq r-p$ and $0 \leq j \leq s-q$. A square $m \times m$ sub-array Y is said to be a *prefix* of X, if Y occurs at position $X[0..m-1, 0..m-1]$. Similarly, Y is said to a *suffix* of X if, Y occurs at position $X[r-m..r-1, s-m..s-1]$.

2.2 Definitions and properties of partially occluded images

In the following we assume that Σ is a finite alphabet of symbols. We denote the symbol $\# \notin \Sigma$ to be a special symbol called the *background* symbol.

Definition 2.1 Let X be a $r \times s$ array called the *image* over the alphabet Σ and let $\mathcal{O} = \{S_1, \ldots, S_k\}$ be a set of $m \times m$ arrays called the *objects* also over Σ. Then the image X is said to be a *valid image* over $\mathcal{O}$ if and only if $X = Z_i$ for some $1 \leq i \leq rs-1$, where

$$Z_0 = \begin{pmatrix} \# & \cdots & \# \\ \vdots & \ddots & \vdots \\ \# & \cdots & \# \end{pmatrix},$$

$$Z_{i+1} = \left(\begin{array}{c|c|c} \text{pref}(Z_i) & \text{sub}(Z_i) & \text{sub}(Z_i) \\ \hline \text{sub}(Z_i) & S_t & \text{sub}(Z_i) \\ \hline \text{sub}(Z_i) & \text{sub}(Z_i) & \text{suff}(Z_i) \end{array}\right)$$

where $S_t \in \mathcal{O}$ for some $t \in \{1, \ldots, k\}$.

The recurrence equalities defined above are said to be the *substitution rules* and the sequence $Z_0, Z_1, \ldots, Z_i$ is said to be the *generating sequence* of the image X over the set of objects $\mathcal{O} = \{S_1, \ldots, S_k\}$. We now construct such a generating sequence for a partially occluded image in the following example.

Example 2.1 Let $\mathcal{O} = \left\{ S_1 = \begin{pmatrix} c & c \\ d & b \end{pmatrix}, S_2 = \begin{pmatrix} a & b \\ b & c \end{pmatrix}, S_3 = \begin{pmatrix} a & d \\ b & c \end{pmatrix}, S_4 = \begin{pmatrix} b & b \\ c & c \end{pmatrix} \right\}$ be the set of objects and let the image be

$$X = \begin{pmatrix} a & b & a & d \\ b & c & b & c \\ c & a & b & b \\ d & b & c & c \\ a & b & d & b \\ b & c & b & c \end{pmatrix}$$

Then X is a valid image over $\mathcal{O}$ with the following generating sequence:

$$Z_0 = \begin{pmatrix} \# & \# & \# & \# \\ \# & \# & \# & \# \\ \# & \# & \# & \# \\ \# & \# & \# & \# \\ \# & \# & \# & \# \\ \# & \# & \# & \# \end{pmatrix}, Z_1 = \begin{pmatrix} \underline{a} & \underline{b} & \# & \# \\ \underline{b} & \underline{c} & \# & \# \\ \# & \# & \# & \# \\ \# & \# & \# & \# \\ \# & \# & \# & \# \\ \# & \# & \# & \# \end{pmatrix}, Z_2 = \begin{pmatrix} a & b & \# & \# \\ b & c & \# & \# \\ \underline{c} & \underline{c} & \# & \# \\ \underline{d} & \underline{b} & \# & \# \\ \# & \# & \# & \# \\ \# & \# & \# & \# \end{pmatrix},$$

$$Z_3 = \begin{pmatrix} a & b & \# & \# \\ b & c & \# & \# \\ c & \underline{a} & \underline{b} & \# \\ d & \underline{b} & \underline{c} & \# \\ \# & \# & \# & \# \\ \# & \# & \# & \# \end{pmatrix}, Z_4 = \begin{pmatrix} a & b & \# & \# \\ b & c & \# & \# \\ c & a & b & \# \\ d & b & c & \# \\ \underline{a} & \underline{b} & \# & \# \\ \underline{b} & \underline{c} & \# & \# \end{pmatrix}, Z_5 = \begin{pmatrix} a & b & \underline{a} & \underline{d} \\ b & c & \underline{b} & \underline{c} \\ c & a & b & \# \\ d & b & c & \# \\ a & b & \# & \# \\ b & c & \# & \# \end{pmatrix},$$

$$Z_6 = \begin{pmatrix} a & b & a & d \\ b & c & b & c \\ c & a & b & \# \\ d & b & c & \# \\ a & b & \underline{a} & \underline{b} \\ b & c & \underline{b} & \underline{c} \end{pmatrix}, Z_7 = \begin{pmatrix} a & b & a & d \\ b & c & b & c \\ c & a & b & \# \\ d & b & \underline{c} & \underline{c} \\ a & b & \underline{d} & \underline{b} \\ b & c & b & c \end{pmatrix}, Z_8 = \begin{pmatrix} a & b & a & d \\ b & c & b & c \\ c & a & \underline{b} & \underline{b} \\ d & b & \underline{c} & \underline{c} \\ a & b & d & b \\ b & c & b & c \end{pmatrix}.$$

The occurrence of the possibly occluded objects in X are underlined. From this construction, it is obvious that the generating sequence of a partially occluded image may not be unique. The decomposition of X into objects is not unique due to the fact that some objects may be partially or totally occluded by others. For example, since S_2 and S_3 share identical first rows, if the second column of S_2 or S_3 is occluded in the image X then there is no way to differentiate between the two objects.

From the above example we can see that there exists many possible generating sequences for a given image, since it's decomposition is not unique. In fact, it can be shown that the number of distinct generating sequences may be exponential in the size of the input image, see [IS-97]. This fact complicates the design of an iterative algorithm for decomposing or even validating a two-dimensional image since it is imperative not to inspect all possible generating sequences for a given image. The definition of a valid image implies trivially that the objects are contained within the image X. That is, we assume that there is no S_i for all $i \in \{1, \ldots, k\}$ that is "cut off" on the edges of the image.

To analyse in more detail a partially occluded image we need to extend the notion of a prefix in two dimensions, which is not as clear and well defined as in the one dimensional case.

Definition 2.2 Let X be an array of size $r \times s$. Then we define a *row-prefix* of X as a rectangular sub-array of X occurring at positions $X[i_1..i_2, 0..j]$ for some $0 \leq i_1 \leq i_2 \leq r-1$ and $0 \leq j \leq s-1$. If $i_1 = 0$ then the sub-array $X[0..i_2, 0..j]$ is called a *proper row-prefix* of X. Similarly for columns, let a *column-prefix* of X be a rectangular sub-array of X occurring at $X[0..i, j_1..j_2]$ for some $0 \leq i \leq r-1$ and

$0 \leq j_1 \leq j_2 \leq s-1$. If $j_1 = 0$ then we say that the sub-array is a *proper column-prefix* of X. See Figure 1(i) for an example of a row-prefix and a column-prefix of X.

We can define a *row-suffix* and a *column-suffix* in a similar way.

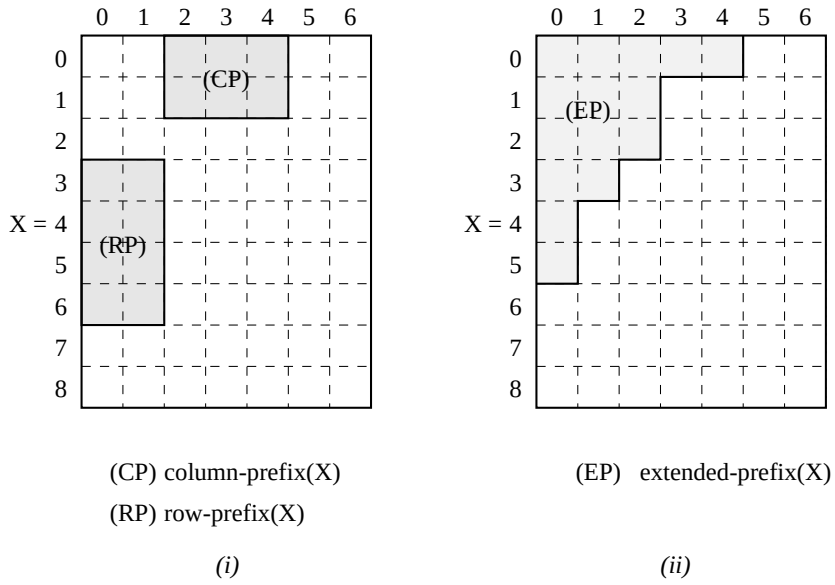

Figure 1: (i) The rectangular sub-array $X[3..6, 0..1]$ is a row-prefix of X and $X[0..1, 2..4]$ is a column-prefix of X. (ii) The staircase like array of points $\mathcal{P} = \{X[0..5, 0..\ell_i - 1] : 0 \leq i \leq 5, \ell_i = [5, 3, 3, 2, 1, 1]\}$ is a decreasing extended-prefix of X since $\ell_i \leq \ell_{i-1}$ $\forall$ $0 \leq i \leq 5$.

This leads to four basic resulting facts if X is a valid image over the set of objects $\mathcal{O} = \{S_1, \ldots, S_k\}$:

Fact 1: For some $i \in \{1..k\}$ there exists a *prefix* of S_i that is also a prefix of X.

Fact 2: For some $i \in \{1..k\}$ there exists a *row-prefix* and a *column-prefix* of S_i occurring at positions $X[l_1..l_2, 0..j]$ and $X[0..l, j_1..j_2]$ respectively with $0 \leq l_1, l_2, l \leq r-1$ and $0 \leq, j_1, j_2, j \leq s-1$.

Fact 3: For some $i \in \{1..k\}$ there exists a *suffix* of S_i that is also a suffix of X.

Fact 4: For some $i \in \{1..k\}$ there exists a *row-suffix* and a *column-suffix* of S_i occurring at positions $X[l_1..l_2, s-j-1..s-1]$ and $X[r-l-1..r-1, j_1..j_2]$ respectively with $0 \leq l_1, l_2, l \leq r-1$ and $0 \leq j_1, j_2, j \leq s-1$.

This follows from the fact that some of the S_i's must occur on the four edges, the top left hand corner and bottom right hand corner of the image X for it to be valid. In an analogue extension from the paper on one-dimensional occluded strings [IS-97], we now break down the validity of a given image into three families of representations of a valid image.

Proposition 2.1 *Let X be an $r \times s$ array over Σ which contains no background symbols #'s. Let $\mathcal{O} = \{S_1, \ldots, S_k\}$ be a set of objects all being $m \times m$ square arrays.*

The array X is a valid image *over $\mathcal{O}$ if and only if one of the following conditions holds:*

$$X = \left(\begin{array}{c|c} \begin{array}{c}\text{proper}\\ \text{pref}(S_i)\end{array} & Y_1 \\ \hline Y_2 & Y_3 \end{array}\right) \tag{7}$$

$$X = \left(\begin{array}{c|c} Y_1 & Y_2 \\ \hline Y_3 & \begin{array}{c}\text{proper}\\ \text{suff}(S_i)\end{array} \end{array}\right) \tag{8}$$

$$X = \left(\begin{array}{c|c|c} Y_1 & Y_2 & Y_3 \\ \hline Y_4 & \text{sub}(S_i) & Y_5 \\ \hline Y_6 & Y_7 & Y_8 \end{array}\right) \tag{9}$$

where the following applies for each equation:

The image resulting from the superposition of an $m \times m$ array of symbols # on top of $properpref(S_i)$, $propersuff(S_i)$ or $sub(S_i)$ together with the resulting sub-arrays Y_j, $j \in \{1,2,3\}$ for equation (7) and (8) and Y_j, $j \in \{1,\ldots,8\}$ for equation (9) must be valid images over $\mathcal{O}$. □

By using the above classification on valid images together with Facts 1 to 4, we aim to achieve a method for efficiently detecting invalid images as a primary task in the design of the algorithm. However before doing so, we need to refine further the notion of a prefix and a suffix of a two-dimensional array.

Definition 2.3 Let X be an array of size $r \times s$. Then we denote by an *extended-prefix* or *staircase prefix* of X a subset of points of X such that:

$$\mathcal{P} = \{X[0..r', 0..\ell_i - 1] : 0 \leq i \leq r' \leq r-1, 0 \leq \ell_i \leq s\}$$

where ℓ_i is either an increasing or decreasing monotone sequence. If $\ell_i \leq \ell_{i-1}$ $\forall i \in \{0..r'\}$ then $\mathcal{P}$ is a decreasing extended-prefix and an increasing extended-prefix otherwise. See Figure 1 (*ii*) for an example of an decreasing extended-prefix of X. If $r' < r-1$ and $\ell_i < s$, $\forall i \in \{0..r'\}$ then we say that $\mathcal{P}$ is a *proper* extended-prefix of X

We can define an *extended-suffix* of X in a symmetrical way.

Following the decomposition that was achieved in Proposition 2.1, we define the validity of a partially occluded image using extended-prefixes.

Proposition 2.2 *Let X be an $r \times s$ image over Σ which contains no background symbols #'s. Let $\mathcal{O} = \{S_1, \ldots, S_k\}$ be a set of $m \times m$ square arrays called the objects. Let $\mathcal{P}(S_j) = \{S[0..m', 0..\ell_i - 1] : 0 \leq i \leq m' \leq m-1, 0 \leq \ell_i \leq m\}$ be a decreasing extended-prefix of some object $S_j \in \mathcal{O}$ occurring at position $X[p,q]$. Then the image X is* valid *over the set of objects $\mathcal{O}$ if and only if the following occurs for any extended-prefix $\mathcal{P}(S_j)$ of X.*

Lets assume first that $\mathcal{P}(S_j)$ is a proper extended-prefix of S_j. Then we claim that every neighbour to the right of the perimeter of the extended-prefix is the occurrence of a row-prefix *or a* column-prefix *of some object in $\mathcal{O}$ for the image X to be valid. If $\mathcal{P}(S_j)$ is a non-proper extended-prefix of S_j then we can only claim that the neighbouring point must be a member of a prefix, a suffix or a substring of some object in $\mathcal{O}$ for the image X to be valid.* □

A similar breakdown can be achieved for increasing extended-prefixes and both increasing and decreasing extended-suffixes.

3 Data Structures and Main Techniques

The algorithm presented here checks the validity of a given partially occluded image according to the definition of a valid image in Definition 2.1. The aim of the algorithm is to decompose the occluded image into a finite set of obstructed objects. If X is valid over the set of objects $\mathcal{O}$ then the algorithm returns a (possible) generating sequence for X, as described in Example 2.1. One can easily extend the case of a rectangular image to that of a square image. In the following, we decompose a square image with a set of square objects. Let X be a $n \times n$ array of symbols drawn from some alphabet Σ called the image. Let a dictionary $\mathcal{O} = \{S_1, \ldots, S_k\}$ consist of a finite set of distinct objects representing $m \times m$ arrays of symbols drawn from Σ.

The main methods used in the algorithm rely upon the computation of the occurrences in X of the longest extended-prefixes and extended-suffixes of the objects in $\mathcal{O}$ and in particular chains of longest extended-prefixes and extended-suffixes in X. In order to achieve this we need to maintain and update several data structures.

Each extended-prefix occurring in X has an associated *prefix-head* at the start of each row and an associated *prefix-tail* at the end of each row. During the iterations of the algorithm we aim to concatenate overlapping extended-prefixes resulting in *extended-prefix-chains*. Every chain is given a head for the first extended-prefix and a tail for the last extended-prefix at every row. All data structures mentioned above also apply in a symmetrical way to deal with suffixes. These data structures will be described in detail in the full version of this paper.

So as to perform queries concerning the longest prefixes of each row of any object occurring in the image X we need to use a trie representing common prefixes.

Definition 3.1 The *common prefix tree* of k strings r_i of length m is a rooted trie (digital search tree) with k leaves such that:

1. Each edge of the tree is labelled with a symbol from the alphabet Σ and is directed away from the root.

2. No two edges emanating from the same node have the same label.

3. Each leaf u is uniquely identified with a string r_i, in the sense that the concatenation of the labels on the path from the root to u is r_i.

4. Each internal node v of height $1 \leq h \leq m-1$ in the tree represents a subset of strings having a common prefix of length h.

3.1 Preprocessing

Step 1:

Construct the Aho-Corasick [AC-75] automata for the rows of each objects in the dictionary $\mathcal{O} = \{S_1, \ldots, S_k\}$. Let $r_i^{(j)} = S_j[i, 0..m-1]$ denote the ith row of the jth object in the dictionary. Let $R = \{r_0^{(1)}, \ldots, r_{m-1}^{(1)}, \ldots\ldots, r_0^{(k)}, \ldots, r_{m-1}^{(k)}\}$.

Building the Aho-Corasick automata for R takes $O(km^2 \log |\Sigma|)$ time, since there are m rows each of length m for each of the k objects in the dictionary and the Aho-Corasick depends on the alphabet Σ.

Step 2:

Construct a *common prefix tree* Γ_i for each of the rows of the objects in the dictionary.

Given a fixed row i, we build Γ_i by refining Γ_{i-1}. First, we set up a first path of length m for the ith row of the first object $S_1[i, 0..m-1]$. Now we do a character comparison for each of the remaining rows by walking down the tree that is being built by querying the automata build in Step 1 and branching out when the symbols are not equal. However the procedure of walking down the tree must be prefix conserving according to Γ_{i-1}.

Once the tree is constructed, we order the internal nodes of the tree by assigning indices to each of internal node. Each such index will represent a subset of objects having a common prefix.

Building a common prefix tree for the k objects and a given row takes $O(km \log k)$ time. To build m such trees (i.e for each row) will take $O(km^2 \log k)$ time.

Step 3:

Preprocess the trees build in Step 2 for answering Lowest Common Ancestor (LCA) queries. By using the algorithm by Harel and Tarjan [HT-84] we can perform this type of query in constant time allowing linear time, in the size of the input, for preprocessing. This will help us answer constant time queries in the prefix tree concerning the longest prefixes of objects.

Step 4:

Create a linked list from the final states in the Aho-Corasick automata of Step 1 pointing to the index of the node they belong to in the associated common prefix tree.

Step 5:

Build a $n \times n$ table $START$, which stores the occurrence of the longest prefix of the *first* row among the objects for each position in X. Initialize a bulletin board of size $n \times n$ corresponding to each position in X. Each position in the bulletin board $START$ stores the following values:

$$START[i,j] = (\ell, \delta), \quad 0 \leq \ell \leq m-1, \quad 1 \leq \delta \leq k,$$

where ℓ represents the length of the *longest prefix* of the *first row* of any object in $\mathcal{O}$. The *unique* identifier δ represents the index found in the prefix tree corresponding to a subset of objects that share a common prefix of length ℓ in their first row. This table of size n^2 can be computed by using the common prefix tree for the first row (Γ_0) of the objects in $O(mn^2 \log k)$ time.

The values computed in the table $START$ identify the starting position of an extended-prefix. After these preprocessing steps we claim that the following can be achieved during the computation of the main algorithm:

Corollary 3.1 *Given the Aho-Corasick automaton $AC(R)$ computed above for all the km rows of the objects the query of testing whether*

$$\text{prefix}_l(X[i, j'..j'+m]) = \text{prefix}_l(r_i^{(j)})$$

requires constant time for a fixed row i.

This fact will allow us to perform $O(m)$ constant time queries for each position in X, yielding a total time complexity of $O(mn^2)$ in the main algorithm.

Step 6:

Do all previous steps (1–5) for dealing in a symmetrical way with suffixes. That is, we reverse all the rows of the objects and compute the AC automata for these rows.

Computing the longest common row prefix trees and preprocessing these for answering LCA queries is straightforward. Then we need to add to the $START$ table the additional suffix values (ℓ, δ), for the longest prefixes of the last row of the objects.

3.2 Main Algorithm and Sub-procedures

The main ideas of the algorithm are outlined below. The algorithm iterates over the points of X by sweeping from left to right over the rows of X. During the iterations we will use the Aho-Corasick automaton and the prefix trees from the preprocessing to answer queries concerning the occurrence of longest row prefixes and suffixes of objects appearing in X. Additionally, we will use a data structure called a "window" of size $2m \times n$ which stores the information of extended-prefixes and extended-suffixes of objects occurring for each position in X. The process of building a decreasing extended-prefix at an arbitrary point $X[p,q]$ with (ℓ, δ) from $START[p,q]$ is done in the following way:

procedure build_extended_prefix$(\ell, \delta, X[p, q..q+m-1])$
begin
$(\ell_0, \delta_0) := (\ell, \delta)$;

$i := 1$;
while $\ell_{i-1} \neq 0$ **do**
$\ell_i :=$ length of longest prefix of the ith row ;
$\delta_i :=$ index of node at height ℓ_i in prefix tree ;
Comment: feed $X[p, q..q + m - 1]$ to the prefix tree Γ_i.
if $\delta_i \neq \delta_{i-1}$ **then** $(\ell_i, \delta_i) \leftarrow LCA(\delta_{i-1}, \delta_i)$;
else $\ell_i := min\{\ell_{i-1}, \ell_i\}$;
$W[i] := \ell_i$;
$i := i + 1$;
return array W and the final δ_i
end

One can extend this construction to the one of building increasing extended-prefixes and extended-suffixes in a symmetrical way.

The aim of this sweeping technique is to create extended-prefixes and extended-suffixes and chain them together to create valid sub-images. For each point that needs to be validated we use the following *decomposition principles* which are based on Proposition 2.2:

(i) The occurrence of an extended-prefix of an object in a valid image must be followed by a (not necessarily proper) row/column prefix of an object.

(ii) If an occurrence of a extended-prefix of an object in an image is followed by an occurrence of a proper extended-suffix of an object, then the image is not valid. In a valid image, the occurrence of a proper extended-suffix of an object is always preceded by the extended-suffix of an object.

(iii) The occurrence of a extended-suffix of an object in a valid image can be followed by either a prefix, a suffix or a proper substring of an object.

(iv) The occurrence of a sub-array of an object in a valid image is preceded and followed by valid images.

3.2.1 Step 0: Initialization of data structures

Initialization: Validate position $X[0, 0]$
begin
Initialize a $2m \times n$ array called the "window" W.
(1) **validate current row** $X[0, 0..m - 1]$.
if $\ell \in START[0, 0]$ **then** $(\ell, \delta) \leftarrow START[0, 0]$;
mark $W[0, 0..\ell - 1]$ with $X[0, 0]$ as *prefix-chain-head*
else return 'image not valid'. Stop.

(2) **Validate next row** $X[1, 0..\ell - 1]$.
$(\ell', \delta') \leftarrow START[1, 0]$;
if $\ell' \in START[1, 0]$ **then** Stop.
Comment: start of a new extended-prefix on next iteration ;
else expand the extended-prefix starting at $X[0, 0]$.
Comment: looking recursively for an extended-prefix down the rows

at most $m-1$ times using procedure build-extended-prefix.

end

3.2.2 Main algorithm

The main steps of the algorithm are as follows:

1. Building chains of extended-prefixes and extended-suffixes using the procedures for augmenting them row by row and the decomposition principles.
2. Creating valid sub-images by concatenating adjacent extended-prefix-chains and extended-suffix-chains.
3. Two-dimensional pattern matching on the remaining blocks of substrings.

The full details of the algorithm will appear in the forthcoming full version of this paper.

4 Conclusion and open problems

The algorithm presented here can be extended to handle variable length objects. An interesting open practical problem is the validation of images with sets of objects that are concave or non-continuous; of particular interest is the variant of the problem with objects over $\Sigma \cup \{\Lambda\}$, where Λ is a transparent symbol and this alphabet defines a set of strings with holes. Another interesting problem is the computation of the *depth* of an object in an image, i.e. the number of objects applied onto an object after the placement of an object in an image. Finally, approximate occlusion analysis is of practical importance and therefore all the above mentioned problems need to be extended to handle errors.

References

[AFI-91] A. Apostolico, M. Farach and C.S. Iliopoulos, Optimal superprimitivity testing for strings, *Information Processing Letters*, (1991), 39, 17–20.

[AC-75] A.V. Aho and M.J. Corasick, Efficient string matching: an aid to bibliographic search, *Comm. ACM*, (1975), 18(6), 333–340.

[BC-94] W. Bischof and T. Caelli, Learning structural descriptions of patterns: a new technique for conditional clustering and rule generation, *Pattern Recognition*, (1994), 27(5), 689–699.

[CIK-98] M. Crochemore, C.S. Iliopoulos and M. Korda, Two-dimensional prefix string matching and covering on square matrices, *Algorithmica*, (1998), 20, 353–372.

[DS-96] A.M. Duval and W.F. Smyth, Covering a circular string with substrings of fixed length, *Int. J. of Foundations of Computer Science*, (1996), 7(1), 87–93.

[HT-84] D. Harel and R.E. Tarjan, Fast algorithms for finding nearest common ancestors, *SIAM J. Comput.*, (1984), 13(2), 338–355.

[IR-97] C.S. Iliopoulos and J.F. Reid, An optimal parallel algorithm for analysing occluded images, In *Proc. 4th Annual Australasian Conference on Parallel And Real-Time Systems*, (1997), University of Newcastle, Australia. N. Sharda and A. Tam (eds), Springer-Verlag, 104–113.

[IS-97] C.S. Iliopoulos and J. Simpson, On-line validation and analysis of occluded images, In *Proc. 8th Australasian Workshop on Combinatorial Algorithms*, (1997), Research on Combinatorial Algorithms, Queensland University of Technology, Australia, V. Estivill-Castro (ed), 25–36.

[KMP-77] D.E. Knuth, J.H. Morris and V.R. Pratt, Fast pattern matching in strings, *SIAM J. Comput.*, (1997), 6, 322-350.

[MS-94] D.W.G. Moore and W.F. Smyth, An optimal algorithm to compute all the covers of a string, *Inform. Process. Lett.*, (1994), 50(5), 239–246.

[MS-95] D.W.G. Moore and W.F. Smyth, A correction to: An optimal algorithm to compute all the covers of a string, *Inform. Process. Lett.*, (1995), 54, 101–103.

[PL-94] P.A. Pevzner and R.J. Lipshutz, Towards DNA sequencing chips, In *Proc. 19th Int. Symp. on Mathematical Foundations of Computer Science*, (1994), Lecture Notes in Computer Science, Springer-Verlag, 841, 143–158.