The Complexity of Two-Dimensional Compressed Pattern-Matching

Piotr Berman [*] **Marek Karpinski** [†] **Lawrence L. Larmore** [‡]
Wojciech Plandowski [§] **Wojciech Rytter** [§]

Abstract

We consider the complexity of problems for highly compressed 2-dimensional texts: *compressed* pattern-matching (when the pattern is not compressed and the text is compressed) and *fully compressed* pattern-matching (when also the pattern is compressed). First we consider 2-dimensional compression in terms of straight-line programs, see [9]. It is a natural way for representing very highly compressed images, by describing larger parts in terms of smaller (earlier described) ones. For 1-dimensional strings there exist polynomial-time deterministic algorithms for similar types of compression [2, 6, 8, 9]. We show that the complexity dramatically increases in a 2-dimensional setting:

- *Compressed matching* for two dimensional compressed images is *NP-complete.*
- *Fully compressed matching* for two dimensional compressed images is $\Sigma_2 P$*-complete.*
- Testing a given occurrence of a two dimensional compressed pattern is *co-NP-complete.*

On the other hand we show efficient algorithms for some related problems:

- Testing equality of two compressed two dimensional patterns (an application of algebraic techniques).
- Testing a given occurrence of an uncompressed pattern in a two dimensional compressed image.

We also show the surprising fact that the compressed size of a subrectangle of a compressed two dimensional array can grow exponentially, unlike the one dimensional case.

[*] Dept. of Computer Science & Eng., Pensylvania State University, University Park, PA16802, USA Email:berman@cse.psu.edu

[†] Dept. of Computer Science, University of Bonn, D-53117, Bonn, Germany. This research was partially supported by the DFG Grant KA 673/4-1 Email:marek@cs.uni-bonn.de

[‡] Department of Computer Science, University of Nevada, Las Vegas, NV 89154-4019. Email:larmore@cs.unlv.edu Research partially supported by National Science Foundation grant CCR-9503441. Part of this work was done while the author was visiting Institut Informatik V, Universität Bonn, Germany.

[§] Instytut Informatyki, Uniwersytet Warszawski, Banacha 2, 02–097 Warszawa, Poland. Supported by the grant KBN 8T11C01208. Email:wojtekpl@mimuw.edu.pl and rytter@mimuw.edu.pl.

1 Introduction

We consider algorithms for problems dealing with *highly compressed* images (two dimensional arrays with entries from some finite alphabet). The objects considered are as much as potentially exponentially compressed. In practice the compression ratio for images can be much larger than in the one dimensional case.
Our main problem is the **Fully Compressed Matching Problem**:

Instance: $Compress(P)$ and $Compress(T)$

Question: does P occur in T?

P is a rectangular *pattern-image* and T is a rectangular *host-image*. The **Compressed Matching Problem** is essentially the same, the only difference being that $P = Compress(P)$, in other words, the pattern is uncompressed. Our results show that any attempt to deal with *highly compressed* (potentially exponentially compressed) two dimensional texts should fail algorithmically. The size of the problem is $n + m$, where $n = |Compress(T)|$ and $m = |Compress(P)|$. Let N determines the total uncompressed size of the problem. Note that in general N can be exponential with respect to n, and any algorithm which decompresses T takes exponential time in the worst case.

We consider also the problems of **Pattern Checking**:

test an occurrence of a pattern at one given position.

This problem has also its *compressed* and *fully compressed* versions.

The first type of compression we consider in this paper is in terms of *straight-line programs* (SLP's), or equivalently, two dimensional context-free grammars generating single objects with the following two operations:

$A \leftarrow BC$, which concatenates images B and C (both of equal height)

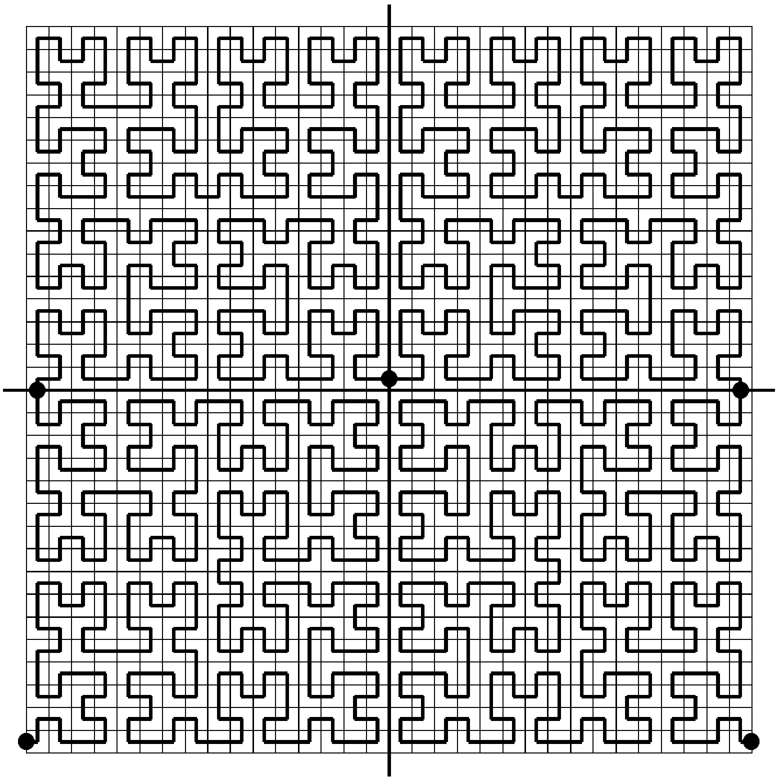
$A \leftarrow B \ominus C$, which puts image B on top of C (both of equal length)

An SLP of size n consists of n statements of the above form, where the result of the last statement is the compressed image. The only constants in our SLP's are letters of the alphabet, interpreted as 1×1 images. We will view SLP's as compressed (descriptions of) images. The complexity of basic string problems for one dimensional texts is polynomial. Surprisingly, the complexity jumps if we pass to two dimensions. The compressed size of a subrectangle of a compressed two dimensional array A can be exponential with respect to the compressed size of A, though such a situation cannot occur in the 1-dimensional case. The proof is omitted in this version.

Theorem 1.1 *For each n there exists an* SLP *describing a text image A_n and a subrectangle B_n of A_n such that the smallest* SLP *describing B_n has exponential size.*

Example 1.

Hilbert's curve can be viewed as an image exponentially compressible in terms of SLP's. An SLP which describes the n^{th} Hilbert's curve, H_n, uses six (terminal) symbols ◱ , ◰ , ◳ , ◲ , ◫ , ⊟ . There are used 12 variables $\square_i$, $\square_i$, $\square_i$, $\square_i$, $\square_i$, $\square_i$, $\square_i$, $\square_i$, $\square_i$, $\square_i$, $\square_i$, $\square_i$, for each $0 \leq i \leq n$. A variable with index i represents a text square of size $2^i \times 2^i$ containing part of a curve. The dots in the boxes show the endpoints of the curve.



$$\square_5 \leftarrow (\ \square_4 \quad \square_4 \ ) \ominus (\ \square_4 \quad \square_4 \ )$$

Figure 1: The 5th Hilbert curve H_5 is composed of four smaller square arrays according to the rule above, it consists of 1024 (terminal) symbols.

The 1×1 text squares are described as follows.

$$\square_0 \leftarrow ◫, \quad \square_0 \leftarrow ◫, \quad \square_0 \leftarrow ⊟, \quad \square_0 \leftarrow ⊟,$$
$$\square_0 \leftarrow ◰, \quad \square_0 \leftarrow ◰, \quad \square_0 \leftarrow ◱, \quad \square_0 \leftarrow ◱,$$
$$\square_0 \leftarrow ◳, \quad \square_0 \leftarrow ◳, \quad \square_0 \leftarrow ◲, \quad \square_0 \leftarrow ◲,$$

The text squares for variables indexed by $i \geq 1$ are rotations of text squares for the variables $\square_i$, $\square_i$, $\square_i$. These variables are composed according to the rules:

$$\square_i \leftarrow \square_{i-1} \ \square_{i-1} \ominus \square_{i-1} \ \square_{i-1} \ ,$$
$$\square_i \leftarrow \square_{i-1} \ \square_{i-1} \ominus \square_{i-1} \ \square_{i-1} \ ,$$
$$\square_i \leftarrow \square_{i-1} \ \square_{i-1} \ominus \square_{i-1} \ \square_{i-1} \ .$$

2 Compressed two dimensional pattern-matching

Recall that the compressed matching problem is to find, given an uncompressed pattern and compressed text (image), whether the pattern occurs within the text.

In our constructions we will use, as a building block, images filled with one kind of letter only, say a. We will use $[a]^i_j$ to denote such $i \times j$ image. It is easy to see that $[a]^i_j$ can be compressed to a SLP of size $O(\log(i) + \log(j))$.

We will use $I, J, \ldots, P, Q, \ldots$ for uncompressed images, and $\mathcal{I}, \mathcal{J}, \ldots, \mathcal{P}, \mathcal{Q}, \ldots$ for compressed ones. Given a compressed image $\mathcal{R}$ (uncompressed image R), we use $\mathcal{R}_{i,j}$ ($R_{i,j}$) to denote the symbol at position (i, j); if the position (i, j) is out of range, we will have $\mathcal{R}_{i,j} = \perp$. We will number the rows and columns starting from 0. We also use the convention that given a number m, $\widetilde{m}$ is a 0-1 vector $(a_0, ..., a_{k-1})$ such that $m = \sum_{i=0}^{k-1} 2^i a_i$. The length of $\widetilde{m}$ should be clear from context. Let $Positions(P) = \{(i, j) : P_{i,j} \neq \perp\}$.

First we consider **Point test problem**: compute the symbol $\mathcal{I}_{i,j}$ for given $\mathcal{I}, i$ and j.

Lemma 2.1 There exists a linear time algorithm for the point test problem.

Theorem 2.2 Compressed matching for two dimensional images is *NP*-complete.

Proof:

To see that the compressed matching is in NP, we can express this problem as the following property of pattern P and image $\mathcal{R}$:

$$\exists (i, j) \{\forall (k, l) \in Positions(P) \;\; P_{k,l} = \mathcal{R}_{i+k,j+l}\}.$$

The equality inside the braces can be tested in polynomial time (Lemma 2.1), hence we have expressed the problem in the normal form for NP.

To show NP hardness, we will use a reduction from 3SAT. Consider a set of clauses $C_0, \ldots, C_{k-1}$, where each clause is a Boolean function of some three variables from the set $\{x_0, \ldots, x_{n-1}\}$. The 3SAT question is whether there exists m such that $C_i(\widetilde{m}) = 1$ for $i = 0, \ldots, k-1$.

Define an $k \times 2^n$ image A as follows: $A_{i,m} = C_i(\widetilde{m})$. Then the 3SAT question is equivalent to the following: does A contain a column consisting of k 1's (i.e. the pattern $[1]^1_m$)? We will reduce 3SAT to the compressed matching problem by showing how to compress A to a SLP with $O(kn)$ statements. Obviously, it suffices to show that we can compress any row of A into a SLP with $O(n)$ statements, because we can combine the compressed rows using $k \ominus$ operations.

Consider then a row R of A corresponding to a clause $C(x_h, x_i, x_j)$ where $h < i < j$. Define $\imath(v_0, \ldots, v_{n-1}) = v_h + 2v_i + 4v_j$, then $R_m = C(\widetilde{\imath(\widetilde{m})})$. We will show how to compress I defined by $I_m = \imath(\widetilde{m})$; then to obtain a SLP for R from SLP $\mathcal{I}$ for I we simply replace each constant a with $C(\widetilde{a})$.

We omit an easy proof of the following fact.

Fact 2.3

$$I \;=\; (((0^{2^h}1^{2^h})^{2^{i-h-1}}(2^{2^h}3^{2^h})^{2^{i-h-1}})^{2^{j-i-1}}((4^{2^h}5^{2^h})^{2^{i-h-1}}(6^{2^h}7^{2^h})^{2^{i-h-1}})^{2^{j-i-1}})^{2^{n-j-1}}$$

To compress I, write a constant length SLP that computes all subexpressions of I, then replace each statement of the form $K \leftarrow L^{2^i}$ with i statements $L \leftarrow LL$ followed by $K \leftarrow L$. One can see that this results in a SLP with $O(n)$ statements. □

3 Fully compressed two dimensional pattern-matching

Recall that the fully compressed matching problem is to find, given a pattern and a text that are both compressed, whether the pattern occurs within the text.

Theorem 3.1 (main result)
Fully compressed matching for two dimensional images is $\Sigma_2 P$-complete.

Given compressed pattern $\mathcal{P}$ and compressed image $\mathcal{I}$, the positive answer to the fully compressed two dimensional pattern matching question is equivalent to the following:

$$\exists(i,j)\forall(k,l) \in \mathit{Positions}(\mathcal{P}) \quad \{\mathcal{P}_{k,l} = \mathcal{I}_{i+k,j+l}\}$$

By Lemma 2.1, the equality in this formula can be checked in polynomial time, hence the problem can be formulated in the normal form of Σ_2^P problems.

This proof of Σ_2^P-hardness requires two lemmas.

Lemma 3.2 *There exists a logspace function f such that for any 3CNF formula F, $f(F) = (u, v, t)$, where u and v are vectors of non-negative integers, t is an integer and*

$$\forall x \quad F(x) \equiv \exists y \; ux + vy = t.$$

Proof: Assume that F has n variables, a clauses with three literals, b clauses with two literals and c clauses with one literal. Vector u will consists of n numbers and vector v of $7a + 3b$ numbers. We will describe each of these numbers, (and t as well) using the identity $\tilde{d} = d^0 \dots d^{(a+b+c-1)}$, where $d^{(k)}$ is *the fragment of d corresponding to clause C_k*. The fragments corresponding to a clause with l literals will have length $2l$. We describe in detail the case of a clause with three literals, the other cases being similar, only simpler.

Assume that clause C_k contains three variables, x_h, x_i, x_j. The fragments of u_h, u_i, and u_j corresponding to C_k are 000100, 000010 and 000001 respectively, while for $l \notin \{h, i, j\}$ we have $u_l^{(k)} = 000000$.

There are 7 truth assignments for (x_h, x_i, x_j) that satisfy $C(k)$, for each one we have an entry in vector v; if v_l is the entry corresponding to a truth assignment (b_0, b_1, b_2) for C_k, then $v_l^{(k)} = 100(1-b_0)(1-b_1)(1-b_2)$. Moreover, for $k' \neq k$ we have $v_l^{(k')} = 0\ldots 0$.

Finally, $t^{(k)} = 100111$.

Consider now x such that $F(x)$ is true. Then the fragment of $\widehat{ux}$ corresponding to a clause C_k is $000b_0b_1b_2$, where (b_0, b_1, b_2) is a truth assignment satisfying C_k (note that x satisfies all the clauses of F). Let v_l be the entry of v corresponding to this truth assignment, and $v_{l_1}, \ldots, v_{l_6}$ be the entries corresponding to other truth assignments that may satisfy C_k. We set y_l to 1 and $y_{l_1}, \ldots, y_{l_6}$ to 0; it is easy to see that the fragment of $\widehat{ux+vy}$ corresponding to C_k is 100111, the same as the corresponding fragment of t. Since this is true for every fragment of t, we have $ux + vy = t$.

Now suppose that there exists y such that $ux+vy = t$. If for every clause C_k exactly one of the entries corresponding to the truth assignments that satisfy C_k has coefficient 1 in the vector y, and if the addition is performed without carries, then each C_k is satisfied. It is easy to prove by induction that this is indeed the case (note that in our string representations of numbers we write the least significant bit first). Details are left to the reader.

Finally, the method of creating (u, v, t) is so regular that it can be carried out by a deterministic log-space Turing machine. □

Define the Σ_2(Subset Sum) problem as follows: given is (u, v, t) where u and v are vectors of positive integers and t is an integer; the question is whether $\exists x \forall y\ ux + vy \neq t$, where the quantifiers range over 0-1 vectors of appropriate length.

Lemma 3.3 *The Σ_2(Subset Sum) problem is Σ_2^P complete.*

Proof: Consider now an arbitrary property L of binary strings that belongs to Σ_2^P. In its normal form, L is represented as

$$L(x) \quad \equiv \quad \exists y_1 \forall y_2\ P(x, y_1, y_2)$$

where P is a polynomial time predicate. Because P⊂NP∩co-NP, the predicate P can be represented as

$$P(x, y_1, y_2) \quad \equiv \quad \neg(\exists y_3 F(x, y_1, y_2, y_3))$$

where F is a 3CNF formula (computed using space which is logarithmic in the size of x in unary). Let "$\cdot$" denote concatenation of vectors. By the previous lemma,

$$F(x, y_1, y_2, y_3) \equiv \exists y_4\ u(x \cdot y_1 \cdot y_2 \cdot y_3) + vy_4 = t$$

where (u, v, t) can be computed in logarithmic space from F. Define $\bar{u}, \bar{v}, \bar{w}$ and $\bar{t}$ so that $u(x \cdot y_1 \cdot y_2 \cdot y_3) + vy_3 = \bar{w}x + \bar{u}y_1\bar{v}(y_2 \cdot y_3 \cdot y_4)$ and $\bar{t} = t - \bar{w}x$. By substitution and De Morgan laws we have,

$$\begin{aligned} L(x) &\equiv \exists y_1 \forall y_2 \neg(\exists y_3 \exists y_4 \; u(x \cdot y_1 \cdot y_2 \cdot y_3) + v y_4 = t) \\ &\equiv \exists y_1 \forall y_2 \forall y_3 \forall y_4 \; u(x \cdot y_1 \cdot y_2 \cdot y_3) + v y_4 \neq t \\ &\equiv \exists y_1 \forall y_2 \forall y_3 \forall y_4 \; \bar{w} x + \bar{u} y_1 \bar{v}(y_2 \cdot y_3 \cdot y_4) \neq t \\ &\equiv \exists y_1 \forall (y_2 \cdot y_3 \cdot y_4) \; \bar{u} y_1 \bar{v}(y_2 \cdot y_3 \cdot y_4) \neq \bar{t} \end{aligned}$$

Because the last of the above statements is an instance of Σ_2(Subset Sum), we have shown that L can be reduced to Σ_2(Subset Sum). □

To prove that fully compressed two dimensional pattern matching is Σ_2^P complete, it suffices to show how to translate an instance of Σ_2(Subset Sum). Consider an instance given by (u, v, t). Recall the definition of T^w from our proof of co-NP completeness. Let U be the image T^u and let V be the image T^v with all row reversed. Recall that dimensions of U and V are $2^n \times (1+r)$ and $2^m \times (1+s)$ respectively, where m and n are the lengths of u and v, while r and s are their sums. We define the pattern and the test as follows:

$$\begin{aligned} &P \leftarrow 1 \ominus [0]^1_{2^n+2^m} \\ &S_0 \leftarrow [0]^{2^n}_{s-t} U \\ &S_1 \leftarrow V[1]^{2^m}_{r-t} \\ &S_2 \leftarrow [0]^{2^n}_{1+r+s-t} \\ &T \leftarrow R_1 \ominus R_2 \ominus R_2 \end{aligned}$$

The subimages S_i's are *stripes* of the text T. Observe first that T contains P if and only if there exists a column of T, say c, that contains P. Because the length of P equals the sum of heights of S_1 and S_2 plus 1, P can start anywhere in the upper stripe S_0 but only there. Because P starts with 1, it must start within U, so $c = s - t + a$ for some $a \geq 0$. Therefore column c consists of column a of U, column $s - t + a$ of V and zeros at the bottom—we can easily exlude the case when this column crosses the middle stripe S_1 through the subimage consisting of 1's only.

Now, column a of U is column a of T^u, so a 1 exists in this column if and only if for some $x < 2^n$ we have $u\tilde{x} = a$. Moreover, column $s - t + a$ of V is column $s - (s - t - a) = t - a$ of T^v, we have all 0's in this column if and only if $vy \neq t - a$ for every $y < 2^m$. Summarizing, P occurs in T if and only if there exists x with the following property: for $a = ux$ the equality $vy = t - a \equiv a + vy = t \equiv ux + vy = t$ holds for no y. Therefore the positive answer to our pattern matching problem is equivalent to the positive answer to the original Σ_2(Subset Sum) problem. This concludes the proof of Theorem 3.1.

4 Fully compressed pattern checking

The problem of fully compressed pattern checking at a given location is to check, given pattern $\mathcal{P}$ and text $\mathcal{R}$ that are both compressed and a position within the text, whether $\mathcal{P}$ occurs within $\mathcal{R}$ at this particular place.

Theorem 4.1 *Fully compressed pattern checking for two dimensional images is co-NP-complete.*

Proof: We can use Lemma 2.1 to express this problem in the normal form of co-NP:

$$\forall (k,l) \in \mathit{Positions}(\mathcal{P}) \;\; \mathcal{P}_{k,l} = \mathcal{R}_{k+i,l+j}.$$

To prove co-NP hardness, we will reduce co-(Subset Sum) to our problem. An instance of co-(Subset Sum) is a vector of integer weights $w = (w_0, \ldots, w_{n-1})$ and a target integer value t; the question is whether $\forall m < 2^n \; w\tilde{m} \neq t$. (Here $w\tilde{m}$ stands for the inner product; because $\tilde{m}$ is a 0-1 vector, $w\tilde{m}$ is a sum of a subset of the terms of w.) We can transform this question to a pattern checking question in a natural manner. Let $s = 1 + \sum_{i=0}^{n-1} w_i$, and let the image T^w consists of 0's and 1's, with $T^w_{m,i} = 1$ if and only if $w\tilde{m} = i$. Then our co-(Subset Sum) question is whether column t of T^w consists of 0's only. In terms of the pattern checking problem, we specify the text T^w, the pattern $[0]^1_{2^n}$ and the position $(t, 0)$.

To finish the proof, we need to compress T^w. Observe that row m of T^w contains exactly one 1, at position $w\tilde{m}$. Moreover, for $m < 2^{n-1}$ we have $w(\widetilde{m + 2^{n-1}}) = w(\tilde{m}) + w_{n-1}$. Therefore when we split T^w into upper and lower halves (each with 2^{n-1} rows), the pattern of 1's is very similar, the only difference being that in the lower half (with higher row numbers) the 1's are shifted by w_{n-1} to the right. Moreover, if we remove the last w_{n-1} zeros from each row in the upper half, we obtain $T^{w(n-1)}$, an image defined just as T^w, but where $w(n-1) = (w_0, \ldots, w_{n-2})$. Applying this observation inductively, we can compute T^w as follows:

```
$T^{w(0)} \leftarrow 1$
for $i \leftarrow 0$ to $n-1$ do
    $U \leftarrow T^{w(i)}[0]^{2^i}_{w_i}$;  $L \leftarrow [0]^{2^i}_{w_i} T^{w(i)}$;  $T^{w(i+1)} \leftarrow U \ominus L$
```

To obtain SLP for T^w we combine $3n+1$ statements of the above program with SLP's that compute auxiliary images $[0]^{2^i}_{w_i}$. It is easy to see that the number of statements in the resulting SLP is $O(n^2 + b)$, where b is the total number of bits in the binary representations of the numbers in vector w. □

5 Equality testing

We reduce equality of two images A and B to equality of two polynomials $Poly(A)$ and $Poly(B)$. The following basic theorem is a version of theorems given by Schwartz and (independently) by Zippel [13].

Theorem 5.1 (nonzero-test theorem)
Let $\mathcal{P}$ be a nonzero polynomial of degree at most d. Assume that we assign to each variable in $\mathcal{P}$ a random value from a set Ω of integers of cardinality R. Then

$$Prob\{\mathcal{P}(\bar{x}) \neq 0\ \} \geq 1 - \tfrac{d}{R}.$$

We can assume that the symbols are integers in some small range, depending on the alphabet. For an $n \times n$ image Z define its corresponding polynomial

$$Poly(Z) = \textstyle\sum_{i,j=1}^{n} Z_{i,j} x^i y^j.$$

Theorem 5.2 There exists a linear time randomized algorithm for testing whether two SLP's compute the same image.

Proof: Observe that two images are equal if and only if their corresponding polynomials are identical. Hence the equality of two images is reduced to testing whether $Poly(A) - Poly(B)$ is identically zero. This can be done efficiently by a randomized algorithm due to Theorem 5.1 and the following fact:

Fact 5.3 *Let $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ be images corresponding to variables A, B, C in some SLP.*

- *If $A \leftarrow B \ominus C$ then $Poly(\mathcal{A}) = Poly(\mathcal{C}) \cdot x^{height(\mathcal{B})} + Poly(\mathcal{B})$.*
- *If $A \leftarrow BC$ then $Poly(\mathcal{A}) = Poly(\mathcal{C}) \cdot y^{width(\mathcal{B})} + Poly(\mathcal{B})$.*

□

6 Compressed pattern checking

Recall that the compressed pattern checking problem is to check whether an uncompressed pattern P occurs at a position (x, y) of an image T given by an SLP $\mathcal{T}$. Let n be the size of $\mathcal{T}$ and N be the size of T. The compressed pattern checking problem can be solved easily in polynomial time by using an algorithm for point test problem $m \cdot k$ times. By Lemma 2.1 there is an algorithm which solves the compressed pattern checking problem in $O(n|P|)$ time. We improve that by replacing n by $\log N \log m$. This is similar to the approach of [6]. If the text image is not very highly compressed then $\log(N)$ is close to $\log(n)$. The idea behind the algorithm is to consider point tests in groups, each group called a *query*. Denote by $\mathcal{V}$ a text which is generated by a variable V. A query is a triple $(V, p, \mathcal{R})$ where V is a variable in SLP $\mathcal{T}$, p is a position inside $\mathcal{V}$ and $\mathcal{R}$ is a subrectangle of the pattern

P. Denote by $\mathcal{R}'$ the rectangle of the same shape as the rectangle $\mathcal{R}$ which is placed at position p in $\mathcal{V}$. We require that $\mathcal{R}'$ abut one of the sides of the rectangle $\mathcal{V}$. An *answer* for a query is true or false depending on whether or not $\mathcal{R}' = \mathcal{R}$. The queries are answered by replacing them by equivalent "simpler" queries. We say that a query $(V, p, \mathcal{R})$ is *simpler* than a query $(V', p', \mathcal{R}')$ if and only if $|\mathcal{V}| < |\mathcal{V}'|$. A query which contains a variable V is called a *V-query*. An *atomic query* is a query $(V, p, \mathcal{R})$ such that $\mathcal{V}$ is a 1×1 square. Clearly, an atomic query can be answered in $O(1)$ time.

The queries are divided into three classes: *strip queries*, *edge queries*, and *corner queries*. Let $(V, p, \mathcal{R})$ be a query. Denote by $\mathcal{R}'$ a rectangle of the same shape as $\mathcal{R}$ and which is positioned at p in $\mathcal{V}$. Then $(V, p, \mathcal{R})$ is a corner query if $\mathcal{R}$ contains at least one side of the pattern or $\mathcal{R}$ is a corner subrectangle of the pattern and $\mathcal{R}'$ is a corner subrectangle of $\mathcal{V}$. The query $(V, p, \mathcal{R})$ is an edge query if $\mathcal{R}'$ contains one side of $\mathcal{V}$. There are four types of edge queries depending on which side of $\mathcal{V}$ is contained in $\mathcal{R}'$. They are called *down*, *left*, *right* and *up* queries. The query $(V, p, \mathcal{R})$ is a strip query if $\mathcal{R}$ is a strip of the pattern and $\mathcal{R}'$ is a strip of $\mathcal{V}$.

The algorithm for the checking problem uses two procedures, *Remove_Edge_Queries*(V, Q) and *Split*(V, Q) where V is a variable in $\mathcal{T}$ and Q is a set of queries. The scheme of the algorithm looks as follows.

```
Algorithm CHECKING
{ input: an SLP $\mathcal{T}$, a pattern $P$ and a position $p$ }
{ output: true iff $P$ occurs at $p$ in a text described by $\mathcal{T}$ }
begin
  $V_1, V_2, \ldots, V_n$:= sort variables in $\mathcal{T}$ on the sizes of their texts, in descending order
  $Q$:={$(V_1, p, P)$}
  for $i$:=1 to $n$ do
      $Q$:=Remove_Edge_Queries($V_i, Q$)  $Q$:=Split($V_i, Q$)
      {there are now only atomic queries in $Q$}  answer all atomic queries in $Q$
end
```

The procedure *Compress_Edge_Queries*(V, Q) deals only with edge *V-queries* in Q. Its aim is to eliminate, for each type of edge query separately, all edge *V-queries* except the query which contains the largest subrectangle of the pattern. We describe how this procedure works for left-edge queries. Let $(V, (0, 0), \mathcal{R})$ be a left-edge query and $\mathcal{R}$ be of maximal size among all left-edge *V-queries* in Q. Let $(V, (0, 0), \mathcal{R}')$ be any other left-edge *V-query*. Then rectangle of shape $\mathcal{R}'$ positioned at $(0, 0)$ in $\mathcal{V}$ is a subrectangle of the rectangle of shape $\mathcal{R}$ positioned at $(0, 0)$ in $\mathcal{V}$. Hence, to answer both queries it is enough to answer the query $(V, (0, 0), \mathcal{R})$ and to check whether the text $\mathcal{R}'$ occurs in $\mathcal{R}$ at $(0, 0)$. Before removing each edge query equality of appropriate rectangles is checked and if the rectangles do not match then the procedure stops and the algorithm returns false.

Assume that $A{:=}BC$ or $A{:=}B \ominus C$ is an assignment for A. The *Split*(A, Q) procedure replaces *A-queries* in Q by equivalent *B-queries* and *C-queries.* Let $(A, p, \mathcal{R})$ be an *A-query* in Q. Consider a rectangle R of shape $\mathcal{R}$ positioned at p in $\mathcal{A}$. Then division of A into B and C according to the assignment for A causes that either R to be wholly contained in B, or wholly in C, or to be divided into two smaller rectangles one of which is in B and the other in C. In the latter case the split of a query is called a *division* of the query.

Fact 6.1 *The total number of all divisions of queries during the work of the algorithm is exactly* $|P| - 1$.

For each variable, edge and corner queries are stored in a list. The data structure for storing strip queries is more sophisticated. For each variable it is a 2-3-tree [1] in which keys are positions of strip rectangles in the variable. Recall that 2-3 trees provide operations *split* and *join* in $O(\log s)$ time where s is the number of elements in the tree.

Fact 6.2 *In each step of algorithm CHECKING the set Q contains at most four corner queries and m strip queries.*

Implementation of the *Split* operation, if it is not a division, requires merging 2-3 trees and this may result in a large number of splits of 2-3 trees. Fortunatelly, it is possible to prove, using arguments similar to those of [6], to prove the following lemma.

Lemma 6.3 *The number of splits of 2-3 trees in algorithm CHECKING is* $O(m \log N)$.

Theorem 6.4 *The algorithm CHECKING works in* $O(|P| + n + (m \log N)(\log m))$ *time.*

Proof: By Fact 6.1, the total cost of all divisions is $O(|P|)$. Total cost of all *Split*s which are not divisions is determined by the number of all corner queries and all edge queries which survive after *Remove_Edge_Queries* operation during the execution of the algorithm and the number of splits of 2-3 trees. This gives, by Lemma 6.3, $O(n + (m \log N)(\log m))$. □

Open Problem: we designed a fast randomized algorithm for the equality of two compressed images, and we conjecture that there is a polynomial time deterministic algorithm for this problem.

References

[1] A.V. Aho, J.E. Hopcroft, and J.D. Ullman, *The design and analysis of computer algorithms*, Addison-Wesley, Reading, Mass., 1974.

[2] A. Amir, G. Benson and M. Farach, *Let sleeping files lie: pattern-matching in Z-compressed files,* in *SODA'94.*

[3] A. Amir, G. Benson, *Efficient two dimensional compressed matching, Proc. of the 2nd IEEE Data Compression Conference* 279-288 (1992).

[4] A. Amir, G. Benson and M. Farach, *Optimal two-dimensional compressed matching,* in *ICALP'94* pp.215-225.

[5] M. Crochemore and W. Rytter, *Text Algorithms,* Oxford University Press, New York (1994).

[6] M. Farach and M. Thorup, *String matching in Lempel-Ziv compressed strings,* in STOC'95, pp. 703-712.

[7] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman (1979).

[8] L. Gąsieniec, M. Karpiński, W. Plandowski and W. Rytter, *Efficient Algorithms for Compressed Strings.* in proceedings of the SWAT'96 (1996).

[9] M. Karpinski, W. Rytter and A. Shinohara, *Pattern-matching for strings with short description,* in *Combinatorial Pattern Matching*, 1995.

[10] D. Knuth, *The Art of Computing, Vol. II: Seminumerical Algorithms. Second edition.* Addison-Wesley, 1981.

[11] A. Lempel and J. Ziv, *On the complexity of finite sequences, IEEE Trans. on Inf. Theory* 22, 75-81 (1976).

[12] A. Lempel and J. Ziv, *Compression of two-dimensional images sequences, Combinatorial algorithms on words* (ed. A. Apostolico, Z.Galil) Springer Verlag (1985) 141-156.

[13] R. Motwani, P. Raghavan, Randomized algorithms, Cambridge University Press 1995.

[14] W. Plandowski, *Testing equivalence of morphisms on context-free languages,* ESA'94, Lecture Notes in Computer Science 855, Springer-Verlag, 460–470 (1994).

[15] J. Storer, *Data compression: methods and theory,* Computer Science Press, Rockville, Maryland, 1988.

[Zi] R.E. Zippel, Probabilistic algorithms for sparse polynomials, in EUROSAM 79, Lecture Notes in Comp. Science 72, 216-226 (1979)

[16] J. Ziv and A. Lempel, *A universal algorithm for sequential data compression, IEEE Trans. on Inf. Theory* vo. IT–23(3), 337–343, 1977.