THE PUBLISHING HOUSE
OF THE ROMANIAN ACADEMY

PROCEEDINGS OF THE ROMANIAN ACADEMY, Series A,
Volume 20, Number 3/2019, pp. 311–319

NETWORKS OF PICTURE PROCESSORS WITH CIRCULAR PERMUTATION

Fernando ARROYO[1], Sandra GOMEZ[1], Victor MITRANA[1,2], José Ramón SANCHEZ[1]

[1] Department of Information Systems, University College of Computer Science
Polytechnic University of Madrid, Crta. de Valencia km. 7 – 28031 Madrid, Spain
[2] National Institute for Research and Development of Biological Sciences,
Independentei Bd. 296, Bucharest, Romania
Corresponding author: Victor Mitrana, E-mail: victor.mitrana@upm.es

Abstract. We propose a new variant of network of evolutionary picture processors, where the operations *mask* and *unmask* considered in [4] are replaced by the circular permutation of a row or column on the picture frontier. We propose a solution based on these networks to the picture pattern matching problem that runs in $\mathcal{O}(n+m+kl)$ computational steps, where the pattern is of size (k,l) and the input picture is of size (n,m). We finally discuss how our solution may easily lead to solutions to a few further related problems on pictures.

Key words: picture, picture language, circularly permuting picture processor, network of picture processors with circular permutation, picture pattern matching.

1. INTRODUCTION

Several generating mechanisms defining picture languages have been introduced and studied in the literature. In [14,15,16,18], pictures are considered as two-dimensional matrices that can be generated by different types of grammars. Other picture processing models have been proposed by the 70's [13], while a hierarchy of generative models for such languages was considered in [17]. Further models extend concepts and results defined for string languages and formal series to two dimensional languages, see, e.g. [8], and to picture series, see, e.g., [6,11]. A good and concise survey is [7]. This work is a continuation of [5], where networks of evolutionary picture processors (NEPP) accepting rectangular pictures have been introduced. Such a network consists of nodes hosting picture processors. A picture processor may either substitute a letter by another letter either in a row or in a column, or delete either a row or a column. The row/column in which the substitution is to take place as well as the row/column which is to be deleted is specified. If the letter to be substituted appears more than once in the specified row/column, then each such occurrence is substituted in different copies of that picture. Therefore, we implicitly assume that sufficiently many copies of every picture are available. All the nodes simultaneously send their pictures to (and receive pictures from) the nodes they are connected to. This process is regulated by input and output filters which allow the pictures to enter and go out from the nodes, respectively.

In [4], two operations, similar to "zoom-in" and "zoom-out", called *mask* and *unmask* are introduced. The variant introduced in [4] is used to solve the problem of pattern matching of a given pattern of size (k,l) in an arbitrary given rectangular picture of size (n,m). The proposed solution is efficient (it runs in $\mathcal{O}(n+m+kl)$ computational steps) and can be extended at no further cost with respect to the number of computational steps to any finite set of patterns, all of them of the same size.

In this paper, we replace the two operations introduced in [4] by just one operation, namely the circular permutation. Thus, the leftmost column, rightmost column, upmost row and downmost row can be shifted as the rightmost column, leftmost column, downmost row and upmost row, respectively. We propose a solution based on the new variant of NEPP to the 2D pattern matching problem which is of the same complexity as that proposed in [4]. Actually, we follow the same strategy to that used in [4]. We finally discuss how our solution may easily lead to solutions to a few further related problems on pictures.

2. BASIC DEFINITIONS

We use [7] for the basic concepts and notations concerning two-dimensional languages. We denote the set of the first n positive integers by $[n]$, while the power set of the set A is denoted by 2^A. The cardinality of a finite set A is denoted by $card(A)$. A *picture* (or a two-dimensional string) over the alphabet V is a two-dimensional array of elements from V. The set of all pictures over the alphabet V is denoted by V_*^*, while a two-dimensional language over V is a subset of V_*^*.

The minimal alphabet containing all symbols appearing in a picture π is denoted by $alph(\pi)$. Let π be a picture in V_*^*; we denote the number of rows and the number of columns of π by $\underline{\overline{\pi}}$ and $|\pi|$, respectively. The pair $(\underline{\overline{\pi}},|\pi|)$ is called the *size* of the picture π. The size of the empty picture ε is obviously (n,m) with $nm=0$. Note that the empty picture is actually the (equivalence) class of all pictures of size (n,m) with $nm=0$. The set of all pictures of size (m,n) over the alphabet V, where $m,n\geq 1$, is denoted by V_m^n. The symbol placed at the intersection of the i th row with the j th column of the picture π is denoted by $\pi(i,j)$.

We recall informally the row and column concatenation operations between pictures. For a formal definition the reader is referred to [7] or [9]. The row concatenation of two pictures π of size (m,n) and ρ of size (m',n') is denoted by ® and is defined only if $n=n'$. The picture π ® ρ is obtained by adjoining the picture ρ under the last row of π. Analogously one defines the column concatenation denoted by ©. Let V be an alphabet; a rule of the form $a\to b$, with $a,b\in V\cup\{\varepsilon\}$ is called an *evolutionary rule*. We say that a rule $a\to b$ is: a) a *substitution rule* if neither of a and b is ε; b) a *deletion rule* if $a\neq\varepsilon$, $b=\varepsilon$; c) an *insertion rule* if $a=\varepsilon$, $b\neq\varepsilon$. In this paper, we shall ignore insertion rules because we want to process every given picture in a space bounded by the size of that picture. Let $Sub_V=\{a\to b\mid a,b\in V\}$ and $Del_V=\{a\to\varepsilon\mid a\in V\}$. Given a rule σ as above and a picture $\pi\in V_m^n$, we define the following *actions* of σ on π:

- If $\sigma\equiv a\to b\in Sub_V$, then
  - If the first column of π contains an occurrence of a, then $\sigma^{\leftarrow}(\pi)$ is the set of all pictures π' such that the following conditions are satisfied:
    (i) there exists $1\leq i\leq m$ such that $\pi(i,1)=a$ and $\pi'(i,1)=b$,
    (ii) $\pi'(j,l)=\pi(j,l)$ for all $(j,l)\in([m]\times[n])\setminus\{(i,1)\}$.
  - If this column does not contain any occurrence of a, then $\sigma^{\leftarrow}(\pi)=\{\pi\}$.

  Informally, $\sigma^{\leftarrow}(\pi)$ is the set of all pictures that can be obtained from π by replacing an occurrence of a by b in the first (leftmost) column of π. Note that σ is applied to all occurrences of the letter a in the leftmost column of π in different copies of the picture π.

  Similarly, we define $\sigma^{\rightarrow}(\pi)$, $\sigma^{\uparrow}(\pi)$, $\sigma^{\downarrow}(\pi)$, $\sigma^{+}(\pi)$, as the set of all pictures obtained by applying σ to the rightmost column, to the first row, to the last row, and to any column/row of π.
- If $\sigma\equiv a\to\varepsilon\in Del_V$, then
  - $\sigma^{\leftarrow}(\pi)$ is the picture obtained from π by deleting the leftmost column of π, provided that this column contains at least one occurrence of a. If this column does not contain any occurrence of a, then $\sigma^{\leftarrow}(\pi)=\{\pi\}$. Analogously, $\sigma^{\rightarrow}(\pi)$, $\sigma^{\uparrow}(\pi)$, and $\sigma^{\downarrow}(\pi)$ is the picture obtained from π by applying σ to the rightmost column, to the first row, and to the last row of π, respectively. Furthermore,
  - $\sigma^{+}(\pi)$ is the set of pictures obtained from π by deleting an arbitrary column or row containing an occurrence of a from π. If more than one column or row of π contains a, then each such column (row) is removed from different copies of π. If π does not contain any occurrence of a, then $\sigma^{+}(\pi)=\{\pi\}$.

For every rule σ, symbol $\alpha \in \{\leftarrow, \rightarrow, \uparrow, \downarrow, +\}$ and $L \subseteq V_*^*$, we define the α-*action* of σ on L by $\sigma^\alpha(L) = \bigcup_{\pi \in L} \sigma^\alpha(\pi)$. Given a finite set of rules M, we define the α-*action of* M on the picture π and the language L by $M^\alpha(\pi) = \bigcup_{\sigma \in M} \sigma^\alpha(\pi)$ and $M^\alpha(L) = \bigcup_{\pi \in L} M^\alpha(\pi)$, respectively. In what follows, we shall refer to the rewriting operations defined above as *evolutionary picture* operations since they may be viewed as the 2-dimensional extensions of the 1-dimensional evolutionary operations.

We now define a new operation on pictures that could replace the insertion operation defined above and not considered here. Let π be a picture of size (m,n) over V.

- $\curvearrowright(\pi)$ returns the picture obtained from π by shifting its leftmost column after the rightmost column of π. Formally, $\curvearrowright(\pi) = \rho \copyright \theta$, provided that $\pi = \theta \copyright \rho$. Analogously, $\curvearrowleft(\pi)$ returns the picture obtained from π by shifting its rightmost column before the leftmost column of π.

- $\text{↳}(\pi)$ returns the picture obtained from π by shifting its upmost row below the downmost row of π. Formally, $\text{↳}(\pi) = \rho \circledR \theta$, provided that $\pi = \theta \circledR \rho$. Analogously, $\text{↱}(\pi)$ returns the picture obtained from π by shifting its downmost row above the upmost row of π.

For every $\circ \in \{\curvearrowright, \curvearrowleft, \text{↳}, \text{↱}\}$ and $L \subseteq V_*^*$, we define $\circ(L) = \{\circ(\pi) \mid \pi \in L\}$.

For two disjoint subsets P (permitting symbols) and F (forbidding symbols) of an alphabet V and a picture π over V, we consider the following two predicates which we will later use to define two types of filters:

$$\begin{aligned} rc_s(\pi; P, F) &\equiv P \subseteq alph(\pi) \wedge F \cap alph(\pi) = \varnothing, \\ rc_w(\pi; P, F) &\equiv alph(\pi) \cap P \neq \varnothing \wedge F \cap alph(\pi) = \varnothing. \end{aligned}$$

For every picture language $L \subseteq V_*^*$ and $\beta \in \{s, w\}$, $rc_\beta(L, P, F) = \{\pi \in L \mid rc_\beta(\pi; P, F) = \text{true}\}$.

An *evolutionary picture processor* (EPP) over V is a 5-tuple (M, PI, FI, PO, FO), where:

- Either $M \subseteq Sub_V$ or $M \subseteq Del_V$. The set M consists of all the evolutionary rules of the processor. As one can see, a processor is “specialized” for just one type of evolutionary operations.
- $PI, FI \subseteq V$ are the *input* sets of permitting and forbidding symbols associated with the processor, while $PO, FO \subseteq V$ are the *output* sets of permitting and forbidding symbols associated with the processor (with $PI \cap FI = \varnothing$ and $PO \cap FO = \varnothing$).

A *circularly permuting picture processor* (CPPP) over V is a 5-tuple (M, PI, FI, PO, FO), where M belongs to $\{\curvearrowright, \curvearrowleft, \text{↳}, \text{↱}\}$, while the other parameters are identical to those defined above for evolutionary processors. An *accepting network of picture processors with circular permutation* (ANPPPC) is a 9-tuple

$$\Gamma = (V, U, G, N, \alpha, \beta, \underline{In}, \underline{Halt}, \underline{Accept}),$$

where:

- V and U are the input and network alphabet, respectively, $V \subseteq U$.
- $G = (X_G, E_G)$ is an undirected graph without loops with the set of vertices X_G and the set of edges E_G. G is called the *underlying graph* of the network.
- N is a mapping which associates with each node $x \in X_G$ the picture processor
$$N(x) = (M_x, PI_x, FI_x, PO_x, FO_x).$$
- $\alpha : X_G \to \{\leftarrow, \rightarrow, \uparrow, \downarrow, +\}$; $\alpha(x)$ is a partial mapping that gives the action mode of the rules of node x on the pictures existing in that node. Note that if x is a CPPP then $\alpha(x)$ is not defined.
- $\beta : X_G \to \{s, w\}$ defines the type of the *input* and *output* filters of a node. More precisely, for every node $x \in X_G$, the following filters are defined:
$$\text{input filter: } \rho_x(\cdot) = rc_{\beta(x)}(\cdot; PI_x, FI_x),$$
$$\text{output filter: } \tau_x(\cdot) = rc_{\beta(x)}(\cdot; PO_x, FO_x).$$
That is, $\rho_x(\pi)$ (resp. $\tau_x(\pi)$) indicates whether or not the picture π can pass the input (resp. output) filter of x. More generally, $\rho_x(L)$ (resp. $\tau_x(L)$) is the set of pictures of L that can pass the input (resp. output) filter of x.
- $\underline{In}, \underline{Halt}, \underline{Accept} \in X_G$ are the *input* node, the *halting* node, and the *accepting* node of Γ, respectively. Of course, it is not obligatory that the three nodes are different from each other.

We then say that $card(X_G)$ is the size of Γ. A *configuration* of an ANPPPC Γ as above is a mapping $C : X_G \to 2^{U_*^*}$ which associates a finite set of pictures with every node of the graph. A configuration may be understood as the sets of pictures which are present in any node at a given moment. Given a picture $\pi \in V_*^*$, the initial configuration of Γ on π is defined by $C_0^{(\pi)}(\underline{In}) = \{\pi\}$ and $C_0^{(\pi)}(x) = \varnothing$ for all $x \in X_G \setminus \{\underline{In}\}$.

Configurations are changed by alternating *processing steps* and *communication steps*. When a configuration C is changed by a processing step, each component $C(x)$ is changed in accordance with the set of rules M_x associated with the node x and the way of applying these rules, namely $\alpha(x)$. Formally, we say that the configuration C' is obtained in *one processing step* from the configuration C', written as $C \Rightarrow C'$, iff $C'(x) = M_x^{\alpha(x)}(C(x))$ for all $x \in X_G$. When changing via a communication step, each node processor $x \in X_G$ sends one copy of each picture it has, which is able to pass the output filter of x, to all the node processors connected to x, and receives all the pictures sent by any node processor connected with x provided that they can pass its input filter. Formally, we say that the configuration C' is obtained in *one communication step* from configuration C, written as $C \vdash C'$, iff

$$C'(x) = \left(C(x) \setminus \tau_x(C(x))\right) \cup \bigcup_{\{x,y\} \in E_G} \left(\tau_y(C(y)) \cap \rho_x(C(y))\right) \text{ for all } x \in X_G.$$

Note that pictures that cannot pass the output filter of a node remain in that node and can be further modified in the subsequent evolutionary steps, while pictures that can pass the output filter of a node are expelled. Further, all the expelled pictures that cannot pass the input filter of any node are lost.

Let Γ be an ANPPPC; the computation of Γ on an input picture $\pi \in V_*^*$ is a sequence of configurations $C_0^{(\pi)}, C_1^{(\pi)}, C_2^{(\pi)}, \ldots$, where $C_0^{(\pi)}$ is the initial configuration of Γ on π, $C_{2i}^{(\pi)} \Rightarrow C_{2i+1}^{(\pi)}$ and

$C_{2i+1}^{(\pi)} \vdash C_{2i+2}^{(\pi)}$, for all $i \geq 0$. Note that configurations are changed by alternative steps. A computation as above *halts* if there exists a configuration such that the set of pictures existing in the halting node is non-empty. The *picture language decided* by Γ is

$$L(\Gamma) = \{\pi \in V_*^* \mid \text{the computation of } \Gamma \text{ on } \pi \text{ halts with a non-empty accepting node}\}.$$

For the rest of this paper, we only deal with ANPPPCs that halt on every input.

3. SOLVING PICTURE MATCHING WITH ANPPPCs

The picture pattern matching problem consists in finding a fixed picture, called pattern, in a given picture. The problem, which is a generalization of the well-known string pattern matching, is motivated by many aspects in the area of low level image processing [12]. A generalization of this problem to pictures that are not two-dimensional arrays is of great interest in the fields of Pattern Recognition, Image Analysis, Computer Vision, etc., see, e.g. [10,19].

Various algorithms exist for the exact two-dimensional matching problem, see, e.g. [3,20] for their time and space complexity. We propose a solution to the two-dimensional pattern matching problem which is based on the networks defined in the previous section. We assume that any picture appears in an arbitrarily large number of identical copies. As our sources of inspiration are some biological phenomena, we consider to be biologically feasible to have sufficiently many identical copies of a molecule. By techniques of genetic engineering, in a linear number of laboratory operations one can get an exponential number of identical 2-dimensional molecules [1,2]. Our construction closely follows the idea developed in [4], hence the first step in our solution is to construct a network able to decide the singleton language formed by a given picture.

PROPOSITION 1. *Let π be a picture of size (k,l), for some $k,l \geq 1$ over an alphabet V. The language $\{\pi\}$ can be decided by an ANPPPC.*

Proof. We define an ANPPPC Γ that decides the singleton language $\{\pi\}$. This network is formed by two disjoint subnetworks. One of these subnetworks checks whether the input picture is exactly π, while the other one makes a computation that is long enough such that the first network can complete its computation. We start by defining the working alphabet U of Γ that consists of the following sets:

$$U_1 = \{[\pi(i,j),i] \mid (i,j) \in ([k-1]\times[l])\}, \quad U_2 = \{\langle \pi(k,j), j\rangle \mid j \in [l]\},$$
$$V' = \{a' \mid a \in V\}, \quad \bar{V} = \{\bar{a} \mid a \in V\}.$$

We now define the nodes of the network deciding π only. We accompany the nodes by some explanations regarding their role. The input node $\underline{In}$ is defined as follows:

M	PI	FI	PO	FO	α	β
$\{a \to a \mid a \in V\} \cup \{a \to a' \mid a \in V\}$	V	$V' \cup \bar{V} \cup U_1 \cup U_2$	$V \cup V'$	$\varnothing$	$+$	w

The input picture, say θ, is initially in the input node $\underline{In}$. Here, in different copies of θ, an occurrence of some a is replaced by a'. The unchanged picture enters X_1 (since it has no prime symbol), while all those having a prime symbol enter Z_1. The nodes X_1 and Z_1 are the starting nodes of two disjoint subnetworks (this means that the two subnetworks never exchange pictures between them). We denote by SN_1 and SN_2 the subnetworks starting with X_1 and Z_1, respectively. The subnetwork SN_1 checks whether the input picture is identical to π, while the subnetwork SN_2 makes a sufficiently long computation allowing the subnetwork SN_1 to complete its computation.

The subnetwork SN_1 is also formed by two subnetworks: a star subnetwork containing the nodes $X_1, X_2, \ldots, X_l$ and $\underline{CP}$ (denoted by SN_{11}), and a path subnetwork containing the nodes $Y_1, Y_2, \ldots, Y_{k-1}$ (denoted by SN_{12}). In the case of $k=1$, SN_{12} is empty. In its turn, the subnetwork SN_2 is a path subnetwork formed by the nodes Z_1, Z_2 and Z_3. The whole underlying network of Γ is depicted in Fig. 1.

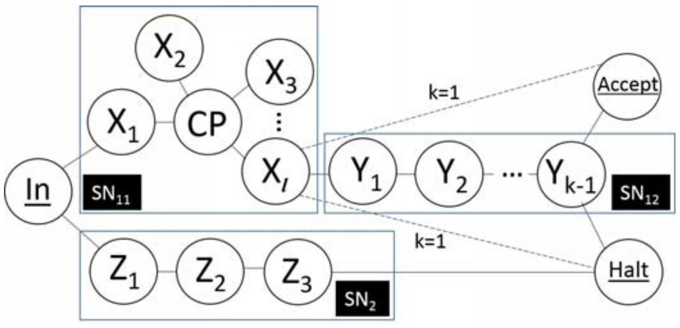

Fig. 1 – The underlying network of Γ.

We continue with the definition of the nodes X_j, $j \in [l]$:

M	PI	FI	PO	FO	α	β
$\{\pi(i,j) \to [\pi(i,j),i] \mid i \in [k-1]\} \cup \{\pi(k,j) \to \langle \pi(k,j),j \rangle\}$	$\{\pi(k,j)\}$	$\{\langle \pi(k,r),r \rangle \mid j \le r \le l\} \cup \begin{cases} V', & \text{if } j=1, \\ \varnothing, & \text{otherwise.} \end{cases}$	$\{\langle \pi(k,j),j \rangle\} \cup \{[\pi(i,j),i] \mid i \in [k-1]\}$	$\varnothing$	$\leftarrow$	s

The definition of $\underline{CP}$ is:

M	PI	FI	PO	FO	β
$\{\curvearrowright\}$	U_2	$\varnothing$	U_2	$\varnothing$	w

We follow the itinerary of the input picture θ that arrives in X_1. In its first column, at least one occurrence of each symbol $\pi(i,1)$, $i \in [k-1]$, is replaced by $[\pi(i,1),i]$. If the first column of θ does not contain $\pi(1,k)$, the picture remain trapped in X_1 forever, because a picture cannot go out from X_1 unless it contains the symbol $\langle \pi(k,1),1 \rangle$. If $\pi(1,k)$ appears in the first column of θ, as soon as one of its occurrences is replaced by $\langle \pi(k,1),1 \rangle$, the new picture leaves X_1. As one can see later in the computation, if a picture leaves X_1 but its first column still contains symbols from V, it will disappear. Let us now follow what happens with a picture, say θ_1, that leaves X_1. Such a picture enters $\underline{CP}$, where a circular permutation takes place such that its first column is moved as the rightmost column. Obviously, the former second column of θ_1 becomes now the leftmost column of the new picture, which enters X_2. The process described above will be applied to all pictures entering X_r and further $\underline{CP}$, for all $2 \le r \le l$. Note that a picture that was processed in a node X_r, for some $r \in [l]$, cannot enter later any node X_j with $j \le r$.

Consequently, a picture that enters $SN_{1,1}$ can leave this subnetwork after visiting exactly once each of the nodes $X_1, X_{i_1}, X_{i_2}, \ldots, X_{i_p}, X_l$ for some $p \ge 0$, and

$$1 < i_1 < i_2 < \ldots < i_p < l, \qquad (*)$$

in this order. Furthermore, the output filters of the nodes X_j, $j \in [l]$, ensures that the input picture has at least k rows. Therefore, the input picture must have at most l columns and at least k rows.

A picture that leaves SN_{11} may enter Y_1, if $k>1$, or $\underline{Halt}$ and $\underline{Accept}$ simultaneously, if $k=1$, provided that it does not contain any symbol from V. For an analysis of this part of the computation when $k>1$, we first define the nodes Y_j, $j\in[k-1]$, of the path subnetwork.

M	PI	FI	PO	FO	α	β
$\{[a,j]\to\varepsilon \mid a\in V$	$\{[a,j]\mid a\in V\}$	$V\cup\begin{cases}\{[a,r]\mid 1\le r<j\}, & \text{if } j\ge 2,\\ \varnothing, & \text{if } j=1.\end{cases}$	$\{\langle\pi(k,r),r\rangle \mid r\in[l]\}$	$\{[a,j]\mid a\in V\}$	$\uparrow$	w

A picture enters Y_1 if it contains at least one symbol $[a,1]$ for some $a\in V$. The first row of all pictures in Y_1 is deleted and the newly obtained pictures are expelled from Y_1. If some of these pictures still contain a symbol $[a,1]$ for some $a\in V$, then they are lost because they can enter neither Y_2 nor X_l.

Assume that a picture ρ_1, obtained by deleting the first row of a picture ρ in Y_1, enters Y_2. This means that $\rho=\mu\circledR\rho_1$, where

$$\mu=([\pi(1,1),1],[\pi(1,i_1),1],\ldots,[\pi(1,i_p),1],[\pi(1,l),1])$$

with $i_1,i_2,\ldots,i_p$ defined in (*). This reasoning may be applied inductively to all pictures that enter $Y_2,Y_3,\ldots,Y_{k-1}$. We now define the halting and accepting node by:

Node	M	PI	FI	PO	FO	α	β
$\underline{Halt}$	$\varnothing$	$\{\langle\pi(k,r),r\rangle \mid r\in[l]\}\cup V'$	$U\setminus PI$	$\{\langle\pi(k,r),r\rangle \mid r\in[l]\}$	$\varnothing$	$+$	w
$\underline{Accept}$	$\varnothing$	$\{\langle\pi(k,r),r\rangle \mid r\in[l]\}$	$U\setminus PI$	$\{\langle\pi(k,r),r\rangle \mid r\in[l]\}$	$\varnothing$	$+$	s

It follows that a picture which enters $SN_{1,2}$ can leave this subnetwork if it has at least k rows, each row j with $j\in[k-1]$ being exactly $([\pi(j,1),j],[\pi(j,i_1),j],\ldots,[\pi(j,i_p),j],[\pi(j,l),j])$ with $i_1,i_2,\ldots,i_p$ defined in (*). After leaving $SN_{1,2}$, a picture that contains more than two rows is lost as it cannot enter any further node. We infer that a picture which enters $SN_{1,2}$ can leave this subnetwork and enter $\underline{Halt}$, provided that it has exactly k rows. The same picture can enter simultaneously $\underline{Accept}$ if it is a row picture with exactly l elements. By all these considerations, we conclude that a picture θ which enters $SN_{1,1}$ can eventually enter simultaneously $\underline{Halt}$ and $\underline{Accept}$ if and only if $\theta=\pi$.

It is worth noting that this computation needs $kl+k+l-1$ processing steps and $kl+k+l-1$ communicating steps to complete.

We now discuss the role of the subnetwork SN_2. Informally, a picture that enters this subnetwork will eventually enter $\underline{Halt}$ after exactly $2kl$ processing steps and $2kl$ communicating steps. The subnetwork SN_2 contains the nodes Z_1,Z_2,Z_3 defined as follows:

Node	M	PI	FI	PO	FO	α	β
Z_1	$\{a\to a'\mid a\in V\}$	V'	$\varnothing$	V'	V	$+$	w
Z_2	$\{a'\to a\mid a\in V\}$	V'	$\varnothing$	V	V'	$+$	w
Z_3	$\{a\to a'\mid a\in V\}$	V	$\varnothing$	V'	$\varnothing$	$+$	w

Indeed, as one can easily see, in Z_1 all symbols $a\in V$ are replaced by a', which takes $kl-1$ processing steps (note that one symbol was replaced in $\underline{In}$), while in Z_2 all the primed symbols are restored, which takes kl processing steps. Finally, before entering $\underline{Halt}$, one more symbol from V is replaced by its primed copy. The proof is complete as soon as we note that $2kl \geq kl + k + l - 1$, which is equivalent to $(k-1)(l-1) \geq 0$.

PROPOSITION 2. *Let* π *be a picture of size* (k,l) *for some* $k,l \geq 1$*, and let* $n \geq k, m \geq l$*. The language*

$$L_{n,m}(\pi) = \{\theta \mid \theta \text{ is a picture of size } (n,m) \text{ and } \pi \text{ is a subpicture of } \theta\}$$

can be decided by an ANPPPC in $\mathcal{O}(n+m+kl)$ *computational (processing and communication) steps.*

Proof. The reasoning here is actually the same as that used in [4]; for sake of completeness, we recall the main considerations from [4]. We give only an informal description of the construction, which is based on a rather simple idea.The network defined in the proof of Proposition 1 will be used as a subnetwork as follows. We add nine new nodes:

– $\underline{In}$, the new initial node that is a substitution node. Here each symbol is replaced by itself.

– Eight new deletion nodes, divided in four pairs of identical nodes: each pair is used for deleting the leftmost column, the rightmost column, the uppermost row and the undermost row, respectively.

Informally, the new nodes cut arbitrary subpictures from the given picture, and send them to the subnetwork constructed in the proof of Proposition 1. Clearly, all subpictures of the same size are produced and sent simultaneously. Further on, as soon as all these pictures leave the new nodes, they cannot further return to them.

All subpictures of the same size received by the subnetwork are matched against the pattern π in parallel. For a given picture is of size (m,n), all subpictures of the same size (k',l') are extracted from the input picture and sent to the subnetwork after exactly $(m-k')+(n-l')+1$ processing steps. If at least one of these subpictures is identical to π, both halting and accepting node will eventually be non-empty after $m-k+n-l+4l$ processing steps. In this case, the input picture is accepted. If the halting node is empty after $m-k+n-l+4l$ steps, it will definitely become non-empty after $m+n+4l-1$ processing steps. As $m+n+4l-1 > m-k+n-l+4l$, the input picture is rejected.

It is easy to note that the construction described above can be extended by ANPPPCs able to detect any pattern from a finite set of pictures, all of them of the same size, at no further computational complexity cost. It suffices to construct an independent subnetwork of the type just discussed for each pattern. This leads to the main result:

THEOREM 1. *Given a finite set* F *of patterns of size* (k,l) *and* (l,k) *for any* $k,l \geq 1$*, the pattern matching problem with patterns from* F *can be solved by ANPPPCs in* $\mathcal{O}(n+m+kl)$ *computational (processing and communication) steps.*

4. CONCLUSIONS

We have proposed an $\mathcal{O}(n+m+kl)$ computational steps solution to the picture pattern matching problem based on ANPPPCs. This construction has a series of consequences. First, it is immediate that any (k,l)-local language [7] with arbitrary k,l can be decided in $\mathcal{O}(n+m+kl)$ computational steps by ANPPPCs. We now define a scattered subpicture π, of size (k,l), in a picture θ, of size (n,m), as being the picture obtained from θ by deleting $n-k$ arbitrary rows and $m-l$ arbitrary columns. The scattered picture pattern matching problem is to find a pattern as a scattered subpicture in a given picture. Our construction for solving the picture pattern matching problem can be easily modified such that:

THEOREM 2. *Given a finite set* F *of patterns of size* (k,l) *and* (l,k)*, for any* $k,l \geq 1$*, the scattered pattern matching problem with patterns from* F *can be solved by ANPPPCs in* $\mathcal{O}(n+m+kl)$ computational *(processing and communication) steps.*

We also define the Hamming distance between two pictures of the same size as the number of positions at which the corresponding symbols are different. The construction in Theorem 1 may be extended to prove:

THEOREM 3. *Let $p \geq 1$ and F be a set of patterns of size (k,l) and (l,k), for any $k,l \geq 1$. For any given picture θ of size (n,m), the problem of finding a pattern in θ which is at a Hamming distance at most p from a picture in F can be solved by ANPPPCs in $\mathcal{O}(n+m+kl)$ computational (processing and communication) steps.*

ACKNOWLEDGEMENTS

Work supported by a grant of the Romanian National Authority for Scientific Research and Innovation, project number POC P-37-257.

REFERENCES

1. L.M. ADLEMAN, Q. CHENG, A. GOEL, M. HUANG, *Running time and program size for self-assembled squares*, Proc. 33rd ACM STOC, pp. 740-748, 2001.
2. G. AGGARWALQ. CHENG, M.H. GOLDWASSER, M.Y. KAO, P.M. DE ESPANES, R.T. SCHWELLER, *Complexities for generalized models of self-assembly*, SIAM Journal on Computing, **34**, *6*, pp. 1493-1515, 2005.
3. A. AMIR, G. BENSON, M. FARACH, *Alphabet independent two dimensional matching*, Proc. 24th ACM STOC, pp. 59-68, 1992.
4. H. BORDIHN, P. BOTTONI, A. LABELLA, V. MITRANA, *Networks of picture processors as problem solvers*, Soft Computing, **21**, *19*, pp. 5529-5541, 2017.
5. P. BOTTONI, A. LABELLA, V. MITRANA, *Networks of evolutionary picture processors*, Fundamenta Informaticae, **131**, pp. 337-349, 2014.
6. S. BOZAPALIDIS, A. GRAMMATIKOPOULOU, *Recognizable picture series*, J. of Automata, Languages and Combinatorics, **10**, pp. 159-183, 2005.
7. D. GIAMMARRESI, A. RESTIVO, *Two-dimensional languages*, in: *Handbook of Formal Languages*, Springer-Verlag, Berlin, pp. 215-267, 1997.
8. D. GIAMMARRESI, A. RESTIVO, *Recognizable picture languages*, Int. J. Pattern Recognition and Artificial Intelligence, **6**, pp. 241-256, 1992.
9. I. INOUE, I. TAKANAMI, *A survey of two-dimensional automata theory*, Proc. 5th Int. Meeting of Young Computer Scientists, Springer-Verlag, Berlin, pp. 72-91, 1990.
10. K. MARRIOTT, B.E. MEYER, *Visual Language Theory*, Springer, 1998.
11. I. MAÜRER, *Characterizations of recognizable picture series*, Theoretical Computer Science, **374**, pp. 214-228**,** 2007.
12. A. ROSENFELD, A.C. KAK, *Digital picture processing*, Academic Press, New York, 1982.
13. A. ROSENFELD, R. SIROMONEY, *Picture languages – a survey*, Languages of design, **1**, pp. 229-245, 1993.
14. G. SIROMONEY, R. SIROMONEY, K. KRITHIVASAN, *Abstract families of matrices and picture languages*, Computer Graphics and Image Processing, **1**, pp. 284-307, 1972.
15. G. SIROMONEY, R. SIROMONEY, K. KRITHIVASAN, *Picture languages with array rewriting rules*, Information and Control, **22**, pp. 447-470, 1973.
16. K.G. SUBRAMANIAN, R. SIROMONEY, *On array grammars and languages*, Cybernetics and Systems, **18**, pp. 77-98, 1987.
17. P.S. WANG, *Hierarchical structure and complexities of parallel isometric patterns*, IEEE Trans. PAM I, **5**, pp. 92-99, 1983.
18. P.S. WANG, *Sequential/parallel matrix array languages*, Journal of Cybernetics, **5**, pp. 19-36, 1975.
19. P.S. WANG, H. BUNKE (eds.), *Handbook on optical character recognition and document image analysis*, World Scientific, 1996.
20. R.F. ZHU, T. TAKAOKA, *A technique for two-dimensional pattern matching*, Communications of the ACM, **32**, pp. 1110-1120 1989.

Received May 12, 2019