Virus Machines And Their Matrix Representations

Antonio Ramírez-de-Arellano[1,2], Francis George C. Cabarle[1,2,3], David Orellana-Martín[1,2], Henry N. Adorna[3], Mario J. Pérez-Jiménez[1,2]

[1]Research Group on Natural Computing, Department of Computer Science and Artificial Intelligence, Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
[2]SCORE lab, I3US, Universidad de Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: aramirezdearellano@us.es, dorellana@us.es, fcabarle@us.es, marper@us.es
[3]Department of Computer Science, University of the Philippines Diliman, 1101 Quezon City, Philippines
E-mail: fccabarle@up.edu.ph, hnadorna@up.edu.ph

Summary. In this work, we present an extension of the matrix representation for virus machines. Structures such as vectors and matrices are useful in practical and theoretical domains. Given the matrix representation of virus machines, the computations of such machines can be expressed in terms of linear algebra operations. Previously, the matrix representation was for deterministic machines only. Presently, we provide a virus machine to nondeterministically generate the set of all natural numbers. We use the virus machine for generating natural numbers to demonstrate the extension of the matrix representation. Finally, we give some conclusions and directions for further work.

Keywords: Virus machine, Matrix representation, Natural computing.

1 Introduction

Virus machines (in short, VMs) are unconventional and bio-inspired models of computing introduced in [1], inspired by the transmission of viruses in a network of hosts. Since their introduction, VMs have been shown to be Turing complete, that is, they can generate, accept, or compute functions over computable sets of numbers [1, 2]. Arithmetic and pairing functions, as well as simulations of workflow patterns have been investigated in the context of VMs [3, 4, 5].

Briefly, a VM is a heterogeneous graph consisting of 3 subgraphs: *host* graph, containing *hosts as nodes*, with *directed and weighted edges* among hosts as *channels*, where hosts contain zero or more copies of the virus object v; *instruction* graph, where *nodes are instructions* to be activated, with directed and

weighted edges between nodes identify which instruction to next activate; *channel-instruction* graph, which connects one instruction node to at most one channel between hosts. By default all channels between hosts are closed: if an instruction i node is activated and i connects to a channel, the channel is opened and viruses are transferred from the source to the destination host.

Previously in [6] a matrix representation for VMs was introduced with the following idea, at some time instant t: the *configuration*, that is, the number of virus objects in each host is represented by a vector; an instruction vector represents the instruction activated at t; a virus transmission matrix of the VM dictates the effect of each instruction (written as rows) to each host (written as columns). Computations of the VM, that is, transitions from one configuration to the next, are provided by linear algebra operations of such a representation.

Other bio-inspired models with matrix representations are spiking neural P systems (SN P systems, in short) [7, 8], including other variants and optimisations in [9, 10, 11, 12]. The matrix representations of SN P systems are used in their automatic design, simulations, and verifications, see for instance [13, 11, 14, 15].

The present work extends the matrix representation of virus machines from [6]. A new VM Π_{Nat} for generating the set of natural numbers is presented. The matrix representation from [6] applies to deterministic VMs only, while the present work extends it to nondeterministic VMs. The VM Π_{Nat} is used to demonstrate the extension with the nondeterministic semantic.

The present work is organised as follows. Section 2 provides the basic definition, syntax, and semantics of VMs. A VM to perform addition, and a new and nondeterministic VM Π_{Nat} are included in Section 2. Section 3 defines the matrix representation first for deterministic VMs, followed by the extensions of the representation for nondeterministic VMs. Conclusions and ideas for further work are provided in Section 4.

2 Virus Machines: Brief definition

In [1], Virus Machines were introduced as a universal model of computation, in the sense that they can calculate every set computable by a Turing machine. While the formal definition can be followed in the founding work, we remark on the syntax here, while the semantics will be explained with an explicit example.

First, let us formally define the **syntax** of virus machines.

Definition 1. *Let a virus machine Π of degree (p, q) with $p, q \geq 0$ defined as:*

$$\Pi = (\Gamma, H, I, D_H, D_H, G_C, n_1, \ldots, n_p, i_1, h_{out})$$

where:

- $\Gamma = \{v\}$ *is the singleton alphabet.*

- $H = \{h_1, \ldots, h_p\}$ *is the ordered set of hosts,* h_{out} *can be either in* H *or not (for this work, we will suppose always* $h_{out} \notin H$*,* $I = \{i_1, \ldots, i_q\}$ *the ordered set of instructions.*
- $D_H = (H \cup \{h_{out}\}, E_H, w_H)$ *is the weighted and directed (WD) host graph, where the edges are called channels and* $w_H : H \times H \cup \{h_{out}\} \to \mathbb{N}$*.*
- $D_I = (I, E_I, w_I)$ *is the WD instruction graph and* $w_I : I \times I \to \{1, 2\}$*.*
- $G_C = (E_H \cup I, E_I)$ *is a unweighted bipartite graph called channel-instruction graph, where the partition associated is* $\{E_H \cup I\}$*.*
- $n_1, \ldots, n_p \in \mathbb{N}$ *are the initial number of viruses in each host* $h_1, \ldots, h_p$*, respectively.*

Regarding the **semantics**, a *configuration* or an *instantaneous description* at an instant $t \geq 0$ is the tuple $\mathcal{C}_t = (a_{1,t}, a_{2,t}, \ldots, a_{p,t}, u_t, a_{0,t})$ where for each $j \in \{1, \ldots, p\}$, $a_{j,t} \in \mathbb{N}$ represents the number of viruses in the host h_j at instant t, and $u_t \in I \cup \{\#\}$. To clarify the notation, in this work it will be said as instantaneous description and $\mathcal{C}_t$ will be noted as ID_t, being $ID_0 = (n_1, \ldots, n_p, i_1, 0)$ the *initial instantaneous description.*

From an instantaneous description ID_t, ID_{t+1} is obtained as follows. The instruction that will be activated is u_t if $u_t \in I$, otherwise ID_t is a *halting configuration.* Let us suppose that $u_t \in I$ and that it is attached to the channel $(h_j, h_{j'}) \in E_H$ with weight $w \in \mathbb{N}$, then the channel is *opened* and two possibilities holds:

- If $a_{j,t} > 0$, then there is *virus transmission*, that is, one virus is consumed from h_j and is sent to the host $h_{j'}$ replicated by w. The next activated instruction will follow the highest weight path in the instruction graph. In case the highest path is not unique, it is chosen nondeterministically. In case there is no possible path, then $u_{t+1} = \#$
- If $a_{j,t} = 0$, then there is no virus transmission and the next instruction follows the least weight path. For the other cases, it is analogous to the previous assumption.

To clarify the behavior of these devices, we will show two specific examples. The first one will be deterministic, and the other one will be nondeterministic. These two examples will be reused for the following section. Before this, some brief explanations of the function computing and number generating modes are presented. For a more formal and detailed definition we refer to [4, 2].

2.1 Virus Machines computing functions: Addition function

A *virus machine with input of degree* (p, q, r) with $p, q \geq 1$ and $1 \leq r \leq p$ is defined as:

$$\Pi = (\Gamma, H, H_r, I, D_H, D_H, G_C, n_1, \ldots, n_p, i_1, h_{out}),$$

where $\Pi = (\Gamma, H, I, D_H, D_H, G_C, n_1, \ldots, n_p, i_1, h_{out})$ is a VM of degree (p, q), and $H_r \subseteq H$ is the ordered set of input hosts. For a given input $(a_1, \ldots, a_r) \in \mathbb{N}$,

the initial configuration of $\Pi + (a_1, \ldots, a_r)$ will be the addition to the input hosts the values $a_1, \ldots, a_r$ respectively.

We say a partial function $f : -\mathbb{N}^r \to \mathbb{N}$ is computed by a VM with input of degree (p, q, r) if for each input $\vec{a} \in \mathbb{N}$ well defined in f, all the computations of $\Pi + \vec{a}$ halt and returns $f(\vec{a})$, otherwise all the computations are non-halting computations.

To clarify this, let Π_{add} be a VM [4] with input of degree $(2, 3, 2)$ visually represented in Figure 1. The hosts are drawn as squares, and instructions are drawn as blue dots; the initial amount of viruses at each host is written inside them. For simplicity, the weights of the arcs with weight 1 are omitted. Finally, the instruction-channel graph is represented by red dotted lines.

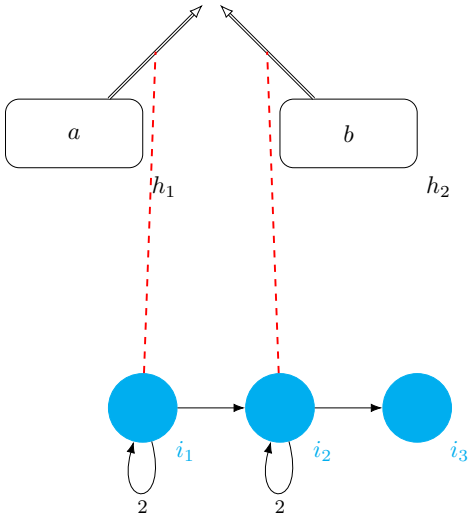

Fig. 1. The VM $\Pi_{Add} + (a, b)$.

A method to formally verify these devices is by looking for **invariants** that highlight relevant loops of the device. For example, two invariants holds in this machine:

$$\varphi(k) \equiv \mathcal{C}_k = (a - k, b, i_1, k), \text{ for each } 0 \leq k \leq a;$$
$$\varphi'(k) \equiv \mathcal{C}_{k+a+1} = (0, b - k, i_2, a + k), \text{ for each } 0 \leq k \leq b;$$

The first invariant φ shows that the a viruses are sent one by one from h_1 to the environment, in this whole process instruction i_1 will be activated, as $\varphi(a)$ is true, the configuration $\mathcal{C}_a = (a - a, b, i_1, a)$ is reached, due to the host h_1 being empty, the next instruction follows the least weight path, that is instruction i_2. Leading the configuration $\mathcal{C}_{a+1} = (0, b, i_2, a)$. At that instant, the second invariant $\varphi'(k)$ is

initialized, which represents the analogous computation, but with host h_2 instead of host h_1. In particular, $\varphi'(b)$ is true, then the following computation holds:

$$\begin{aligned}\varphi'(b) =& \mathcal{C}_{a+b+1} = (0, b-b, i_2, a+b),\\ & \mathcal{C}_{a+b+2} = (0, 0, i_3, a+b), \text{ as } h_2(0),\\ & \mathcal{C}_{a+b+3} = (0, 0, \#, a+b).\end{aligned}$$

Thus, after the $a+b+3$ transition steps, the machine halts and returns $a+b$, which is the addition between (a, b).

2.2 Virus machines generating sets: Natural numbers set

VMs can be defined to generate sets of natural numbers; we say a number $n \in \mathbb{N}$ is generated by a VM Π, if for a computation of Π the machine returns n. We say that a subset $A \subseteq \mathbb{N}$ is generated by a virus machine Π if and only if for every $a \in A$, the number a is generated by Π, and for any halting computation of Π, the output belongs to the set A.

To clarify this, let Π_{nat} be the VM of degree $(2, 4)$ depicted in Figure 2 that generates the set $\mathbb{N} \setminus \{0\}$.

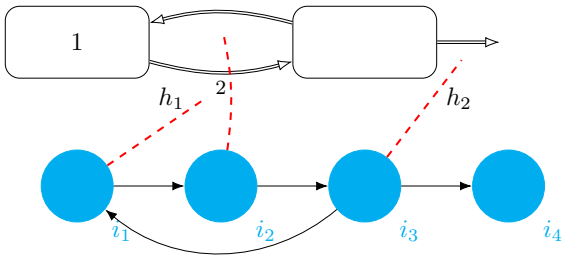

Fig. 2. The VM Π_{Nat}.

To formally prove that the natural numbers set $\mathbb{N}$ is generated, let us see that for each $n \in \mathbb{N} \setminus \{0\}$, there exists a computation of Π_{nat} that generates n. Let us suppose that the number generated is $n \in \mathbb{N} \setminus \{0\}$, then the invariant that holds this machine is:

$$\varphi''(k) \equiv \mathcal{C}_{3k} = (1, 0, i_1, k), \text{ for each } 0 \leq k \leq n-1;$$

The invariant can be easily proved by induction. In particular $\varphi(n)$ is true, thus the following computation holds:

$$\begin{aligned}\varphi''(n-1) \equiv & \mathcal{C}_{3(n-1)} = (1, 0, i_1, n-1),\\ & \mathcal{C}_{3(n-1)+1} = (0, 2, i_2, n-1),\\ & \mathcal{C}_{3(n-1)+2} = (1, 1, i_3, n-1),\\ & \mathcal{C}_{3n} = (1, 0, i_4, n), \text{ nondeterministic decision},\\ & \mathcal{C}_{3n+1} = (1, 0, \#, n),\end{aligned}$$

Thus, after $3n + 1$ transition steps, the machine halts and returns n.

To demonstrate that each halting computation of Π_{nat} is in $\mathbb{N} \setminus \{0\}$, we only have to prove that for any halting computation of Π_{nat}, the output is greater than zero. For this, let us highlight the fact that at the instant $t = 3$, only two possible computations arise: $\mathcal{C}_3 = (1, 0, i_1, 1)$ or $\mathcal{C}_3 = (1, 0, i_4, 1)$. In both cases, one virus has been sent to the environment, as it cannot decrease, the output will be greater than zero. Thus, VM Π_{nat} generates the set $\mathbb{N} \setminus \{0\}$.

3 Matrix Representation

In this section the matrix representation is formally defined for deterministic virus machines, after that, the first ideas on the matrix representation of the nondeterministic virus machines are presented.

3.1 Determinism case

Regarding the semantics of a VM, for any step or instant $t \geq 0$, the *instantaneous description* of Π is $ID_t = (a_{1,t}, a_{2,t}, \ldots, a_{p,t}, u_t, a_{0,t})$, where each $a_{i,t}$ is the number of viruses in the host h_i, the instruction u_t is next activated, and the environment contains $a_{0,t}$ viruses.

For the following definitions, consider a VM Π of degree (p, q), with the notation fixed above and in Definition 1, at any instant t of its computation. We note that definitions and results in the present section for the deterministic case are from [6].

Definition 2 ([6]). *We define the* **configuration vector** *as the vector*

$$\overrightarrow{c}_t = \langle a_{1,t}, a_{2,t}, \ldots, a_{0,t} \rangle,$$

and the **instruction vector** *as the vector*

$$\overrightarrow{i}_t = \langle r_{1,t}, r_{2,t}, \ldots, r_{q,t} \rangle,$$

where $r_{m,t} = 1$ if $u_t = i_m \in I$, otherwise $r_{m,t} = 0$, for $1 \leq m \leq p$. That is, if the activated instruction is $i_j \in I$, then the component $r_{j,t}$ is the only non-zero element of $\overrightarrow{i}_t$.
In particular, vector $\overrightarrow{c}_0 = \langle n_1, n_2, \ldots, n_p\ 0 \rangle$, and $\overrightarrow{i}_0 = \langle 1, 0, \ldots, 0 \rangle$, is the **initial configuration vector** *and* **initial instruction vector**, *respectively.*

Definition 3 ([6]). *A* **virus transmission matrix** *of Π is defined as*

$$M_{\Pi} = [m_{k,j}]_{q \times p+1},$$

where

$$m_{k,j} = \begin{cases} -1, & \text{if instruction } i_k \text{ activates to remove a virus from host } h_j, \\ w, & \text{if } h_j \text{ (or the environment) receives } w \text{ viruses when } i_k \text{ activates}, \\ 0, & \text{otherwise.} \end{cases}$$

Let us apply Definition 2 and Definition 3, to the deterministic Π_{Add} in Figure 1 of Section 2.1. We have $\overrightarrow{c}_0 = \langle a, b, 0 \rangle$ and $\overrightarrow{i}_0 = \langle 1, 0, 0 \rangle$, to mean the following: hosts h_1 and h_2 have a and b viruses, respectively, and the environment is empty, with instruction i_1 first activated. The virus transmission matrix $M_{\Pi_{Add}}$ of Π_{Add} is given by Equation 1.

$$M_{\Pi_{Add}} = \begin{pmatrix} -1 & 0 & 1 \\ 0 & -1 & 1 \\ 0 & 0 & 0 \end{pmatrix} \tag{1}$$

The idea of the virus transmission matrix $M_{\Pi_{sub}}$ is to show the effects of the instructions (the rows) to the hosts and environment (the columns). For instance, row 1 of $M_{\Pi_{Add}}$ shows that i_1 has no effect (hence the 0 element) on column 2 (for h_2). The effect of i_1 is to remove 1 and add 1 virus each to h_1 and the environment, respectively. Similarly, row 2 shows that i_2 removes and adds one virus to h_2 and the environment, respectively, but has no effect on h_1. Lastly, i_3 leads to halting as no other instructions follow it.

Definition 4 ([6]). *The* ***instruction control matrices*** *are matrices defined as*

$$M_{I,1} = [a_{k,j}]_{q,q}, \text{ and } M_{I,2} = [b_{k,j}]_{q,q},$$

where

$$a_{k,j} = \begin{cases} 1, & \text{if } (i_k, i_j) \in E_I \text{ and } w_I((i_k, i_j)) = 1, \\ 0, & \text{otherwise.} \end{cases}$$

$$b_{k,j} = \begin{cases} 1, & \text{if } (i_k, i_j) \in E_I \text{ and } w_I((i_k, i_j)) = 2, \\ 0, & \text{otherwise.} \end{cases}$$

To obtain the **configuration transition equation** and the **instruction transition equation**, we need to compute some partial results. Depending on the existence or not of a virus in the origin host, the next configuration is changed or not, respectively.

Define the following *partial configuration* vectors

$$\overrightarrow{c}'_t = \overrightarrow{i}_t \cdot M_{\Pi}, \overrightarrow{c}''_t = \overrightarrow{c}_t + \overrightarrow{c}'_t, \overrightarrow{c}'''_t = \overrightarrow{c}''_t I_{p+1 \times p+2} \tag{2}$$

where $I_{p+1 \times p+2}$ is the identity matrix with $p+1$ rows and $p+2$ columns, since there is one column more than rows, the last column is filled with zeros. The idea

behind each vector is: $\overrightarrow{c}'_t$ is the vector to be subtracted from $\overrightarrow{c}_t$ if there was a virus in the origin host. $\overrightarrow{c}''_t$ is the result of the subtraction between $\overrightarrow{c}_t$ and $\overrightarrow{c}'_t$. If there exists a -1, then it means that there were no viruses present in the origin host. We extend the vector $\overrightarrow{c}''_t$ with one more zero in the vector $\overrightarrow{c}'''_t$ for the following technical detail: Let $m_t = min(\overrightarrow{c}'''_t)$ the **control coefficient at instant t**, then m_t is 0 if there was at least one virus in the origin host and -1 otherwise. To obtain the next configuration vector we have the following result.

Theorem 1 ([6]). *Let Π be a VM with q instructions and p hosts, M_Π is the virus transmission matrix, $\overrightarrow{c}_t$ and $\overrightarrow{i}_t$ are the configuration and instruction vectors at instant t, respectively. We obtain the next configuration vector $\overrightarrow{c}_{t+1}$ using the following* **transition equation***:*

$$\overrightarrow{c}_{t+1} = \overrightarrow{c}_t + (1 + m_t)\overrightarrow{c}'_t.$$

Let us apply Definition 4, Theorem 1 to $M_{\Pi_{Add}}$. Given $\overrightarrow{c}_0 = \langle a, b, 0\rangle$ and $\overrightarrow{i}_0 = \langle 1, 0, 0\rangle$ we have

$$\overrightarrow{c}'_0 = \overrightarrow{i}_0 \cdot M_{\Pi_{sub}} = \langle -1, 0, 1\rangle, \overrightarrow{c}''_0 = \overrightarrow{c}_0 + \overrightarrow{c}'_0 = \langle (a-1), b, 1\rangle,$$

with $\overrightarrow{c}'''_0 = \overrightarrow{c}''_0 \cdot I_{3\times 4} = \langle (a-1), b, 1, 0\rangle$. We also have $m_0 = min((a-1), b, 1, 0, 0)$ so that if $a > 0$ we have $m_0 = 0$. The *next configuration* of Π_{Add} is

$$\overrightarrow{c}_1 = \overrightarrow{c}_0 + (1 + m_0) \cdot \overrightarrow{c}'_0 = \langle a, b, 0\rangle + \overrightarrow{c}'_0 = \langle (a-1), b, 1\rangle.$$

Let us now move to definitions to obtain the equation for the next instruction, using m_t defined above.
Let the *partial instruction* vectors and *control sum* be

$$\overrightarrow{i}_{t,1} = \overrightarrow{i}_t \cdot M_{I,1},\ \overrightarrow{i}_{t,2} = \overrightarrow{i}_t \cdot M_{I,2},\ \overrightarrow{i}'_t = \overrightarrow{i}_{t,1} + 2\overrightarrow{i}_{t,2}, s_t = i'_t \cdot 1_{q\times 1} \quad (3)$$

s_t is the **control sum at instant** t, being $1_{q\times 1}$ a column vector with q ones. s_t is a scalar number that has 4 possible values:

$$s_t = \begin{cases} 0, & \text{if current instruction } \overrightarrow{i}_t \text{ has no next instructions,} \\ 1, & \text{if current instruction } \overrightarrow{i}_t \text{ has one next instruction,} \\ 2, & \text{if current instruction } \overrightarrow{i}_t \text{ has two next instructions,} \\ & \text{both of them with an arc of weight 1,} \\ 3, & \text{if current instruction } \overrightarrow{i}_t \text{ has two next instructions, one with an arc} \\ & \text{of weight 1 and one with an arc of weight 2,} \end{cases} \quad (4)$$

If we *restrict the VM Π to be deterministic* (that is, $s_t \neq 2$), we can define the next instruction i_{t+1} as follows:

$$\begin{aligned}\overrightarrow{i}_{t+1} = \frac{(1-s_t)(2-s_t)(3-s_t)}{6}\overrightarrow{i}_{t,1} + \frac{(-s_t)(2-s_t)(3-s_t)}{-2}\overrightarrow{i}_{t,1} + \\ + \frac{(-s_t)(1-s_t)(2-s_t)}{-6}(m_t \cdot \overrightarrow{i}_{t,1} + (1+m_t)\overrightarrow{i}_{t,2})\end{aligned} \tag{5}$$

If we simplify the terms for $\overrightarrow{i}_{t+1}$, the following result provides the next instruction to be activated.

Theorem 2 ([6]). *Let Π be a VM of degree (p, q), M_Π the virus transmission matrix, $M_{I,1}$ and $M_{I,2}$, the instructions control matrices, $\overrightarrow{c}_t$ and $\overrightarrow{i}_t$ are the configuration and instruction vectors at instant t, respectively. We obtain the next configuration vector $\overrightarrow{i}_{t+1}$ using the following* **instruction control equation***:*

$$\overrightarrow{i}_{t+1} = \frac{2-s_t}{6}(((m_t 2)s_t^2 + (5-m_t)s_t + 3)\overrightarrow{i}_{t,1} + s_t(1-s_t)(1+m_t)\overrightarrow{i}_{t,2}), \tag{6}$$

where $\overrightarrow{i}_{t,j} = M_{I,j}\overrightarrow{i}_t$, for $j \in \{1, 2\}$, m_t and s_t are the control coefficient and the control sum at instant t, respectively.

Remark 1. Theorem 2 is true if and only if Π is deterministic.

Let us apply the partial instruction vectors, s_t, and Theorem 2 to $M_{\Pi_{Add}}$. Now $s_0 = 3$ from equation 4 since the two arcs of i_1 has a sum of 3 for their weights. Due to $s_0 = 3$ only the rightmost term of equation 5 is nonzero, and more specifically the term with $\overrightarrow{i}_{0,2}$, providing $\overrightarrow{i}_1 = 1 \cdot \overrightarrow{i}_{0,2} = \langle 2, 0, 0\rangle$. The *next instruction* to be activated is i_1 again due to $a > 0$ at instant $t = 0$.

3.2 Non-determinism case

As we stated in Remark 1, the definitions and results presented in the previous subsection are for deterministic virus machines, however, this computing paradigm develops nondeterministic computing models, so it should be taken into account. In this subsection we develop the first ideas on this purpose.

First, Theorem 1 remains true for nondeterministic behavior, as the nondeterminism comes from the instruction that will be activated in the following step. Because of this, we will focus on Theorem 2.

For being the nondeterministic case at an instant t, it means that the highest/least weight path is not unique, that is the case $s_t = 2$, where the two possible next instructions have an arc of weight 1. In addition, $\overrightarrow{i}_{t,1}$ has two non-zero components and $\overrightarrow{i}_{t,2}$ is the zero vector. Keeping the same notation as before and applying Theorem 2 the following equation holds:

$$\overrightarrow{i}_{t+1} = \frac{2-s_t}{6}(((m_t 2)s_t^2 + (5-m_t)s_t + 3)\overrightarrow{i}_{t,1} + s_t(1-s_t)(1+m_t)\overrightarrow{i}_{t,2}),$$

Evaluating the variables we have that $\overrightarrow{i}_{t+1}$ is the zero vector. To alleviate this problem, we propose an extension of the equation by adding the following term $\frac{s_t(1-s_t)(3-s_t)}{-2}\overrightarrow{i}_{t,1}$. Thus, the following Theorem holds:

Theorem 3. *Let Π be a VM of degree (p, q), M_Π the virus transmission matrix, $M_{I,1}$ and $M_{I,2}$, the instructions control matrices, $\overrightarrow{c}_t$ and $\overrightarrow{i}_t$ are the configuration and instruction vectors at instant t, respectively. We obtain the next configuration vector $\overrightarrow{i}_{t+1}$ using the following* **auxiliary instruction control equation***:*

$$\begin{aligned}\overrightarrow{i}''_{t+1} = \frac{2-s_t}{6}(((m_t 2)s_t^2 + (5-m_t)s_t + 3)\overrightarrow{i}_{t,1} + s_t(1-s_t)(1+m_t)\overrightarrow{i}_{t,2}) + \\ + \frac{s_t(1-s_t)(3-s_t)}{-2}\overrightarrow{i}_{t,1},\end{aligned} \tag{7}$$

where $\overrightarrow{i}_{t,j} = M_{I,j}\overrightarrow{i}_t$, for $j \in \{1, 2\}$, m_t and s_t are the control coefficient and the control sum at instant t, respectively.

The vector $\overrightarrow{i}''_{t+1}$ is a binary vector that can be written as $\overrightarrow{i}''_{t+1} = \sum_{k\in K} e_k$, where $K \subseteq \{1, \ldots, q\}$, and e_k is the corresponding euclidean basis vector. Then **instruction control equation** *holds:*

$$\overrightarrow{i}_{t+1} = e_{k'},$$

where k' is nondeterministically chosen from the set K.

Let us show the example presented in Subsection 2.2 to clarify this. First, we have $\overrightarrow{c}_0 = \langle 1, 0, 0\rangle$ and $\overrightarrow{i}_0 = \langle 1, 0, 0, 0\rangle$. The virus transmission matrix $\mathcal{M}_{\Pi_{nat}}$ is given by Equation 8.

$$M_{\Pi_{nat}} = \begin{pmatrix} -1 & 2 & 0 \\ 1 & -1 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & 0 \end{pmatrix} \tag{8}$$

As we said in the Subsection 2.2, at instant 3 a nondeterministic decision is made, let us see how it works with the matrix representation. For that, let us see the instant 2, we have $\overrightarrow{i}_2 = \langle 0, 0, 1, 0\rangle$, and $\overrightarrow{c}_2 = \langle 1, 1, 0\rangle$. From here we have the following:

$$\begin{aligned}\overrightarrow{c}'_2 &= \langle 0, -1, 1\rangle, \\ \overrightarrow{c}''_2 &= \langle 1, 0, 1\rangle, \\ \overrightarrow{c}'''_2 &= \langle 1, 0, 1, 0\rangle.\end{aligned}$$

Thus, $m_2 = 0$. By Theorem 1, we have:

$$\overrightarrow{c}_3 = \overrightarrow{c}_2 + (1 + m_t)\overrightarrow{c}_2 = \langle 1, 0, 1\rangle.$$

What is new here is how we obtain the following instruction vector. Here we have the following:

$$\begin{aligned}\overrightarrow{i}_{2,1} &= \langle 1, 0, 0, 1\rangle,\\ \overrightarrow{i}_{2,2} &= \langle 0, 0, 0, 0\rangle,\\ \overrightarrow{i}'_2 &= \langle 1, 0, 0, 1\rangle,\\ s_2 &= 2.\end{aligned}$$

By the Theorem 3 we have $\overrightarrow{i}''_3 = \overrightarrow{i}_{2,1}$, which can be written as $\overrightarrow{i}''_3 = e_1 + e_4 = \langle 1, 0, 0, 0\rangle + \langle 0, 0, 0, 1\rangle$. Thus, a nondeterministic decision arises choosing $\overrightarrow{i}_3 = e_1$ or $\overrightarrow{i}_3 = e_4$. That represents, exactly, the nondeterministic decision of going to i_1 or i_4 as expected.

4 Conclusion

In recent years, transforming or representing computing processes in linear algebra operations has been a major scope because of their efficient implementations. In this work, a matrix representation of virus machines has been presented with two explicit examples, one with only the deterministic behavior, and the other with nondeterministic behavior. It is interesting to note that this representation opens an interesting framework for the invariants, which is crucial in the formal verification of these devices.

A next direction is to apply the representation in this work to the simulation of workflow patterns in [5]. Such patterns have been studied previously in the framework of spiking neural P systems (in short, SN P systems), see for instance [16, 17, 18]. In order to represent VMs for such patterns the representation needs to be extended to (instruction or channel) parallel VMs, another interesting direction. The matrix representation of VMs, perhaps with a corresponding implementation in software, can help in the verification of the simulated patterns.

The matrix representation in [6] and extended in the present work further opens the simulation in massively parallel processors, such as graphics processing units (in short, GPUs). GPUs are also known as *accelerators* due to their more optimised performance with linear algebra structures, compared to CPUs. For instance, many P system simulations benefit from the use of GPUs [19]. More specifically, the matrix representations of SN P systems, see for instance [7], result in further optimisations in their GPU simulations [10, 14]. For instance the ideas from [11] are experimentally validated in [20], with larger and exhaustive tests [21] which outperform state-of-the-art GPU software libraries.

It is also interesting to continue investigating *reachability*, other static or dynamic properties of VMs, and the complexity of deciding such properties. For

instance, given some configuration $\overrightarrow{c}$ is a configuration $\overrightarrow{c}'$ reachable, where $\overrightarrow{c} \neq \overrightarrow{c}'$? That is, is there a sequence of transitions starting from $\overrightarrow{c}$ and ends with $\overrightarrow{c}'$? The matrix representation can help with this problem, as well as other ideas such as liveness, deadness, coverability [22, 23].

Acknowledgements

F.G.C. Cabarle is supported by the QUAL21 008 USE project, "Plan Andaluz de Investigación, Desarrollo e Innovación" (PAIDI) 2020 and "Fondo Europeo de Desarrollo Regional" (FEDER) of the European Union, 2014-2020 funds. A. Ramírez-de-Arellano is supported by the Zhejiang Lab BioBit Program (Grant No. 2022BCF05).

References

1. Chen, X., Pérez-Jiménez, M.J., Valencia-Cabrera, L., Wang, B., Zeng, X.: Computing with viruses. Theoretical Computer Science **623** (2016) 146–159
2. Ramírez-de-Arellano, A., Orellana-Martín, D., Pérez-Jiménez, M.J.: Generating, computing and recognizing with virus machines. Theoretical Computer Science **972** (07 2023) 114077
3. Ramírez-de Arellano, A., Orellana-Martín, D., Pérez-Jiménez, M.J.: Basic arithmetic calculations through virus-based machines. In Ferrández Vicente, J.M., Álvarez-Sánchez, J.R., de la Paz López, F., Adeli, H., eds.: Bio-inspired Systems and Applications: from Robotics to Ambient Intelligence, Cham, Springer International Publishing (2022) 403–412
4. Ramírez-de Arellano, A., Orellana-Martín, D., Pérez-Jiménez, M.J.: Using virus machines to compute pairing functions. International Journal of Neural Systems **33**(05) (2023) 2350023
5. Ramírez-de-Arellano, A., Cabarle, F.G.C., Orellana-Martín, D., Riscos-Núñez, A., Pérez-Jiménez, M.J.: Virus machines at work: Computations of workflow patterns. International Summer Conference (ISC 24) of Decision Science Alliance, 6-7 June 2024 València, Spain
6. Ramírez-de Arellano, A., Cabarle, F.G.C., Orellana-Martín, D., Pérez-Jiménez, M.J., Adorna, H.N.: Matrix representation of virus machines. In Ferrández Vicente, J.M., Val Calvo, M., Adeli, H., eds.: Bioinspired Systems for Translational Applications: From Robotics to Social Engineering, Cham, Springer Nature Switzerland (2024) 420–429
7. Zeng, X., Adorna, H., Martínez-del Amor, M.Á., Pan, L., Pérez-Jiménez, M.J.: Matrix representation of spiking neural P systems. In: Membrane Computing: 11th International Conference, CMC 2010, Jena, Germany, August 24-27, 2010. Revised Selected Papers 11, Springer (2011) 377–391
8. Adorna, H.N.: Matrix representations of spiking neural P systems: Revisited. arXiv preprint arXiv:2211.15156 (2022)

9. Jimenez, Z.B., Cabarle, F.G.C., de la Cruz, R.T.A., Buño, K.C., Adorna, H.N., Hernandez, N.H.S., Zeng, X.: Matrix representation and simulation algorithm of spiking neural p systems with structural plasticity. Journal of Membrane Computing **1** (2019) 145–160
10. Carandang, J.P., Cabarle, F.G.C., Adorna, H.N., Hernandez, N.H.S., Martínez-del Amor, M.Á.: Handling non-determinism in spiking neural P systems: Algorithms and simulations. Fundamenta Informaticae **164**(2-3) (2019) 139–155
11. Martínez-del Amor, M.Á., Orellana-Martín, D., Pérez-Hurtado, I., Cabarle, F.G.C., Adorna, H.N.: Simulation of spiking neural p systems with sparse matrix-vector operations. Processes **9**(4) (2021) 690
12. Ballesteros, K.J., Cailipan, D.P.P., de la Cruz, R.T.A., Cabarle, F.G.C., Adorna, H.N.: Matrix representation and simulation algorithm of numerical spiking neural p systems. Journal of Membrane Computing **4**(1) (2022) 41–55
13. Aboy, B.C.D., Bariring, E.J.A., Carandang, J.P., Cabarle, F.G.C., De La Cruz, R.T., Adorna, H.N., Martínez-del Amor, M.Á.: Optimizations in cusnp simulator for spiking neural p systems on cuda gpus. In: 2019 International Conference on High Performance Computing & Simulation (HPCS), IEEE (2019) 535–542
14. Gungon, R.V., Hernandez, K.K.M., Cabarle, F.G.C., De la Cruz, R.T.A., Adorna, H.N., Martínez-del Amor, M.Á., Orellana-Martín, D., Pérez-Hurtado, I.: Gpu implementation of evolving spiking neural p systems. Neurocomputing **503** (2022) 140–161
15. Gheorghe, M., Lefticaru, R., Konur, S., Niculescu, I.M., Adorna, H.N.: Spiking neural p systems: matrix representation and formal verification. Journal of Membrane Computing **3**(2) (2021) 133–148
16. Cabarle, F.G.C., Adorna, H.N.: On structures and behaviors of spiking neural p systems and petri nets. In: Membrane Computing: 13th International Conference, CMC 2012, Budapest, Hungary, August 28-31, 2012, Revised Selected Papers 13, Springer (2013) 145–160
17. Cabarle, F.G.C., Buño, K.C., Adorna, H.N.: Time after time: Notes on delays in spiking neural p systems. In Nishizaki, S.y., Numao, M., Caro, J., Suarez, M.T., eds.: Theory and Practice of Computation, Tokyo, Springer Japan (2013) 82–92
18. Song, T., Zeng, X., Zheng, P., Jiang, M., Rodriguez-Paton, A.: A parallel workflow pattern modeling using spiking neural p systems with colored spikes. IEEE transactions on nanobioscience **17**(4) (2018) 474–484
19. Martínez-del-Amor, M.A., García-Quismondo, M., Macías-Ramos, L.F., Valencia-Cabrera, L., Riscos-Núñez, A., Pérez-Jiménez, M.J.: Simulating P systems on GPU devices: a survey. Fundamenta Informaticae **136**(3) (2015) 269–284
20. Hernández-Tello, J., Martínez-Del-Amor, M.Á., Orellana-Martín, D., Cabarle, F.G.: Sparse matrix representation of spiking neural p systems on gpus. In Vaszil, G., Zandron, C., Zhang, G., eds.: Proc. International Conference on Membrane Computing (ICMC 2021), Chengdu, China and Debrecen, Hungary, 25 to 26 August 2021 (Online). (2021) 316–322
21. Hernández-Tello, J., Martínez-Del-Amor, M.Á., Orellana-Martín, D., Cabarle, F.G.C.: Sparse spiking neural-like membrane systems on graphics processing units. International Journal of Neural Systems **34**(7) (2024) 2450038–2450038
22. Peterson, J.L.: Petri nets. ACM Computing Surveys (CSUR) **9**(3) (1977) 223–252
23. Cabarle, F.G.C., Adorna, H.N.: On structures and behaviors of spiking neural p systems and petri nets. In: Membrane Computing: 13th International Conference,

CMC 2012, Budapest, Hungary, August 28-31, 2012, Revised Selected Papers 13, Springer (2013) 145–160