Journal of Membrane Computing (2026) 8:56–65
https://doi.org/10.1007/s41965-025-00190-3

RESEARCH PAPER

A solution to SAT with virus machines with pre-computed resources

David Orellana-Martín[1,2] · **Claudio Zandron**[3] · **Alberto Leporati**[3]

Received: 11 November 2024 / Accepted: 14 April 2025 / Published online: 7 May 2025

Abstract
In Natural Computing, different real-life processes can appear as the inspiration for a new model of computation. Virus machines use the spread and replication of biological viruses as an inspiration for a model of computation with three well-differentiated graphs: the *hosts* graph, that acts like the memory; the *instructions* graph, that acts as a program; and the *instructions-channel* graph, that controls the flow of information through the system. In previous works, the computational power and problem-solving capabilities of this model have been demonstrated. In this work, we provide an application for solving the SAT problem in polynomial time using an **EXP**-uniform family of super virus machines with OR channel parallelism.

Keyword Virus machines, SAT problem, EXP-uniform solution

1 Introduction

In the area of Natural Computing, different real-life processes can appear as the inspiration for a new model of computation. A paramount example in this respect are Membrane systems (also known as P systems), first introduced by Păun in [22], that constitute a computational framework inspired by biological cells, functioning in a parallel and distributed way. These systems are noted for their decentralized characteristics, with their evolution established by the rules defined inside different compartments bounded by membranes.

Significant research has been devoted to investigating this model, considering many different aspects. Recent studies have addressed topics such as computational properties [19, 21], efficiency in computation [1, 13, 14], relationships with other formal models, including Petri nets [4], morphogenetic systems [38], and Markov chains [35], as well as applications to real-world problems [3, 7, 24, 39, 45]. Various adaptations of P systems have been proposed and thoroughly examined, such as P systems with active membranes [15, 23, 25, 37], spiking neural P systems [6, 10, 11, 16, 26, 42, 46], tissue P systems [20, 44], and P colonies [5, 12]. Recent research has also focused on simulating P systems using mainstream hardware [2, 41], developing formal verification methods [17, 18], and adopting more visual methodologies [8].

In 2015, virus machines [40] were introduced. Besides being a Turing-complete model, other interesting approaches have demonstrated their computational power [28, 33, 34]. Their operation is inspired upon the way viruses spread among various hosts and replicate within an organism. Such a spread and replication are governed by a specific set of rules, operating on a network of interacting hosts. In the last years, some interesting applications of virus machines have been proposed, such as attacking cryptosystems [27] and modeling power systems [43].

The basic variant of virus machines, while powerful from a computational point of view, is quite inefficient, because of its inherent sequential behavior. In fact, only one instruction is executed at a time, and that instruction can only open one single channel, that will move one virus from one host to another one. To increase the efficiency, some variants have been proposed in the literature, such as virus machines with host excitation [29], stochastic virus machines [31], and more recently, parallel virus machines [30]. In particular,

✉ David Orellana-Martín
dorellana@us.es

[1] Research Group on Natural Computing, Department of Computer Science and Artificial Intelligence, Universidad de Sevilla, Seville, Spain

[2] SCORE Lab, I3US, Universidad de Sevilla, Avda. Reina Mercedes s/n, 41012 Seville, Spain

[3] Dipartimento di Informatica, Sistemistica e Comunicazione, Università degli Studi di Milano-Bicocca, Viale Sarca 336, Edificio U14, 20126 Milan, Italy

in [43] two different semantics are applied while using channel parallelism. On the one hand, if an instruction opening different channels needs a virus to pass through at least one channel to select the path with the highest value, then the virus machine is said to be using the OR semantics. On the other hand, if a virus needs to pass through all channels connected to the instruction, then the virus machine is said to be using the AND semantics. In this work, we combine OR semantics with the superchannels defined in [32], and we present an **EXP**-uniform family of so-called *super virus machines* that solves the SAT problem in linear time with respect to the number of variables and clauses.

The rest of the paper is organized as follows. The next section is devoted to introducing some notions and notations that are needed to make the paper self-contained. In Sect. 3, the new variant of super virus machines combining superchannels with OR semantics will be defined, introducing its computational ingredients. Section 4 will be devoted to defining the **EXP**-uniform family of virus machines that solves SAT, and to give an overview of how the proposed solution works. The paper ends with some conclusions and open research lines for future work.

2 Preliminaries

In this section, we recall some notions, to fix the notation and to make the paper self-contained.

2.1 Basic notions of set theory and formal language theory

Let $\mathbb{N} = \{0, 1, \dots\}$ be the set of natural numbers. An alphabet Σ is a finite and non-empty set of elements, called *symbols*. An alphabet that contains only one symbol is called a *singleton alphabet*.

A multiset over an alphabet Σ is an ordered pair (Σ, f) such that f is a mapping from Σ to $\mathbb{N}$. For $a \in \Sigma$, the value $f(a)$ denotes the *multiplicity* of symbol a in the multiset. A multiset (Σ, f) can be represented as the set $\{a_1^{f(a_1)}, \dots, a_k^{f(a_k)}\}$ or as any permutation of the string $a_1^{f(a_1)} \dots a_k^{f(a_k)}$, with $a_i \in \Sigma$ and $f(a_i) > 0$ for all $1 \leq i \leq k$. If $\Sigma = \{a\}$ is a singleton alphabet, then a multiset can simply be represented as the string $a^{f(a)}$ if $f(a) > 0$, and with the empty string if $f(a) = 0$.

2.2 Propositional Boolean formulas and the SAT problem

A Boolean variable x is a variable that can take two different values, either 1 or 0, that can be interpreted as the logical values *true* and *false*, respectively. A literal is either a Boolean variable x or the negation of a Boolean variable $\neg x$. A clause is a disjunction of literals. A propositional Boolean formula in conjunctive normal form (CNF) is a conjunction of clauses. Let $Var(\varphi) = \{x_1, \dots, x_n\}$ be the set of variables appearing in a propositional Boolean formula φ; then, a truth assignment σ to such a formula φ is a function $\sigma : Var(\varphi) \to \{1, 0\}$ that assigns a truth value to each variable. Once each Boolean variable has been assigned a truth value, clauses and formulas can be evaluated by interpreting the logical operators in the usual way. We thus obtain that a clause is satisfiable if any of its literals is satisfiable, whereas a CNF formula is satisfiable if all of its clauses are satisfiable.

SAT is an **NP**-complete decision problem [9] that can be formulated as follows: given a CNF Boolean formula $\varphi = C_1 \wedge C_2 \wedge \dots \wedge C_m$, where each clause $C_j = X_{j_1} \vee X_{j_2} \vee \dots \vee X_{j_{i_j}}$ is the disjunction of literals $X_{j_1}, X_{j_2}, \dots, X_{j_{i_j}}$ defined over the set of variables $Var(\varphi) = \{x_1, \dots, x_n\}$, determine whether there exists an assignment $\sigma : Var(\varphi) \to \{1, 0\}$ that satisfies φ.

In what follows, we write $x_i \in C_j$ (respectively, $\neg x_i \in C_j$) to indicate that the literal x_i (resp., $\neg x_i$) appears in the clause C_j.

2.3 Cantor pairing function

The Cantor pairing function $\langle \cdot, \cdot \rangle$ is a bijective function from $\mathbb{N}^2$ to $\mathbb{N}$, defined as follows:

$$\langle x, y \rangle = \frac{(x + y) \cdot (x + y + 1)}{2} + y.$$

For each pair of natural numbers x, y, the Cantor pairing function produces a unique natural number, thus making it a good candidate as a size encoding function, that is, a function that expresses the size of a problem instance through a single value.

2.4 EXP-uniform solutions for decision problems

Formally, a decision problem X is a pair (I_X, θ_X) such that I_X is a language over a finite alphabet (whose elements are called instances) and θ_X is a total Boolean function over I_X. In the case of the SAT problem, I_X is the set of strings that describe all possible CNF Boolean formulas, whereas θ_X maps to *true* all such formulas which are satisfiable.

A polynomial encoding (cod, s) of X is a pair of functions that can be computed in polynomial time by deterministic Turing machines, such that for each instance $u \in I_X$, $s(u)$ is a natural number and $cod(u)$ is an encoding of u to be fed as

input to the virus machine $V(s(u))$. For the SAT problem, u will be (a string representing a) CNF Boolean formula containing n variables and m clauses, and we will use the Cantor pairing function $\langle \cdot, \cdot \rangle$ as the function s, so that $s(u)$ will be $\langle n, m \rangle$. The value $cod(u)$ will be an appropriate encoding of the CNF Boolean formula u, to be fed as input to the virus machine $V(s(u)) = V(\langle n, m \rangle)$.

In what follows, we will build uniform families of virus machines. This means that for each possible instance size $s(u)$, the machine $V(s(u))$ will be able to solve all possible instances of size $s(u)$. We will specify the instance we want to solve by giving $cod(u)$ as input to $V(s(u))$, and we will denote the resulting virus machine with the embedded input by $V(s(u)) + cod(u)$.

Precisely, we will use **EXP**-uniform families of virus machines to solve the SAT problem. We say that a family $\mathbf{V} = \{V(n) \mid n \in \mathbb{N}\}$ is an **EXP**-uniform solution for a decision problem $X = (I_X, \theta_X)$ if the following conditions hold:

- The family $\mathbf{V}$ is exponentially uniform by Turing machines; that is, for each $n \in \mathbb{N}$ there exists a deterministic Turing machine that works in exponential time with respect to n and constructs the virus machine $V(n)$ from n.
- There exists a polynomial encoding (cod, s) of X in $\mathbf{V}$ such that:

  1. The family $\mathbf{V}$ is *polynomially bounded* with respect to (X, cod, s); that is, there exists a natural number $k \in \mathbb{N}$ such that for each instance $u \in I_X$, every computation of $V(s(u)) + cod(u)$ takes, at most, $|u|^k$ computation steps.
  2. The family $\mathbf{V}$ is sound and complete with respect to (X, cod, s), that is:

     – We say that the family $\mathbf{V}$ is *sound* with respect to (X, cod, s) if for each instance $u \in I_X$, if *there exists* at least one accepting computation of $V(s(u)) + cod(u)$ then $\theta_X(u) = 1$.
     – We say that the family $\mathbf{V}$ is *complete* with respect to (X, cod, s) if for each instance $u \in I_X$, if $\theta_X(u) = 1$, then all the computations of $V(s(u)) + cod(u)$ are accepting computations.

In this sense, the family $\mathbf{V}$ provides devices loaded with an exponential amount of computational resources. Since these resources are not built during *computation time* but rather while the machine $V(s(u))$ is being built, we say that the systems of such a family $\mathbf{V}$ use *pre-computed resources*; that is, before starting the computation of such devices, a huge (in this case, exponential) amount of resources has been generated and can be used to solve the problem.

3 Super virus machines with OR channel parallelism

In this section, we give the definition of virus machines when superchannels [32] and OR semantics in channels [43] are considered. For simplicity, we will call the resulting model *super virus machines*.

A parallel virus machine with superchannels and OR (channel parallelism) semantics, of degree (p, q), is a tuple

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

where:

1. $\Gamma = \{v\}$ is a singleton alphabet, where the only element is called a *virus*;
2. $H = \{h_1, \ldots, h_p\}$ is the set of *hosts*, $H_i \subseteq H$ is the set of input hosts, and $I = \{i_1, \ldots, i_q\} \cup \{\#\}$ is the set of *control instructions*. All these are ordered sets, such that the following conditions hold: $H \cap I = \emptyset, v \notin H \cup I, h_{out} \notin \Gamma \cup I$ and $i_1 \in I$;
3. $D_H = (H \cup \{h_{out}\}, E_H, w_H)$ is a weighted directed graph, called the *hosts graph*, where $E_H \subseteq H \times (H \cup \{h_{out}\})$ is such that $(h, h) \notin E_H$ for all $h \in H$, *out-degree*$(h_{out}) = 0$ and w_H is a mapping from E_H to $\mathbb{N}\backslash\{0\}$;
4. $D_I = (I, E_I, w_I)$ is a weighted directed graph, called the *control instructions graph*, where $E_I \subseteq I \times I$, *out-degree* $(i) \leq 2$ for all $i \in I$, and w_I is a mapping from E_I to $\mathbb{N}\backslash\{0\}$;
5. $G_C = (V_C, E_C)$ is a directed bipartite graph, called the *instructions-channels graph*, where $V_C = I \cup E_H$, being $\{I, E_H\}$ the associated partition, and $E_C \subseteq V_C \times V_C$;
6. $n_j \in \mathbb{N}$, for $1 \leq j \leq p$, is the number of viruses initially placed in host h_j;
7. $i_1 \in I$ is the *initial control instruction*, that is, the first instruction to be executed at the beginning of the computation;
8. $h_{out} \in H$ is the *output host*, that is, the host in which the output is collected.

A parallel virus machine with superchannels and OR semantics for channel parallelism $V = (\Gamma, H, H_i, I, D_H, D_I, G_C, n_1, \ldots, n_p, i_1, h_{out})$ of degree (p, q) can be seen as a set of p hosts, each host h_j containing initially n_j viruses, and a set of q instructions. The hosts are connected through *channels* that can be open or closed, and are initially closed. E_H is defined as $E_S \cup E_N$, such that $E_S \cap E_N = \emptyset$, where channels from E_N are usual channels whereas channels from E_S are called *superchannels*. As it will become clear in a moment, the difference between the two types of channels is the following: while in a usual

channel only one virus will pass at a time, in a superchannel all the viruses contained in the host located at the incoming end of the channel will transit. In both cases, the number of passing viruses will be multiplied by the weight (a natural number) associated with the (super)channel. The instructions graph marks the flow of the computation, each instruction having the ability to *open* the channels it is attached to.

A *configuration* of a parallel virus machine with superchannels and OR channel parallelism, in a specific moment t, denoted $C_t = (n_{1,t}, \ldots, n_{p,t}, i_t, n_{0,t})$, is given by the number of viruses contained in each host at that moment, the currently *active* instruction i_t, and the number $n_{0,t}$ of viruses in the environment. The initial configuration of V is thus given by $C_0 = (n_1, \ldots, n_p, i_1, 0)$. If an input $m \in \mathbb{N}\backslash\{0\}$ is introduced in a virus machine V, a number of extra viruses will be placed in the corresponding input host; we will denote by $V + m$ the virus machine thus obtained. A configuration is halting if $i_t = \#$.

A *computation step* of a virus machine V, currently in the configuration $C_t = (n_{1,t}, \ldots, n_{p,t}, i_t, n_{0,t})$, will occur as follows. First of all, the instruction i_t will open all the channels it is attached to. From the point of view of the hosts, if a channel (respectively, superchannel) (h_i, h_j) is opened, if $n_{i,t} = 0$ then no viruses will pass through it, whereas if $n_{i,t} > 0$ then one virus (resp., $n_{i,t}$ viruses) will be removed from the initial host and $w_H(h_i, h_j)$ viruses (resp., $w_H(h_i, h_j) \cdot n_{i,t}$ viruses) will be added to the receiving host through the channel (resp., the superchannel). We must take into account that if k channels coming out of the host h_i are opened, then two different scenarios may arise: on the one hand, if $n_{i,t} \geq k$, then a virus will pass through each channel and k viruses will be removed from such a host; on the other hand, if $n_{i,t} < k$, then all the viruses will be removed from that host and the channels where the viruses pass through are selected in a non-deterministic way. In [43] two different semantics are applied while using channel parallelism. On the one hand, if an instruction that opens multiple channels requires a virus to go through at least one channel by selecting the path with the highest value, then the virus machine is said to be using OR semantics. On the other hand, if a virus needs to pass through all channels connected to the instruction, then the virus machine is said to be using the AND semantics. From the point of view of the control instructions, following the OR channel parallelism behavior described in [43], if a virus passes through at least one channel that is opened by an instruction i_j, then the next instruction will be i_k such that $(i_j, i_k) \in E_I$ and there is no $i_m \in I$ such that $(i_j, i_m) \in E_I$ and $w_I(i_j, i_k) < w_I(i_j, i_m)$; that is, the path with the highest value will be selected. Otherwise, if the current instruction is not attached to any channel or if it is attached but there are no viruses to be moved, then the path with the lowest value will be selected; that is, the next instruction will be i_k such that $(i_j, i_k) \in E_I$ and there is no $i_m \in I$ such that $(i_j, i_m) \in E_I$ and $w_I(i_j, i_k) > w_I(i_j, i_m)$. If the out-degree of i_j is 2 and both paths have the same weight, then the followed path will be selected in a non-deterministic way. If the out-degree of i_j is 0 then i_j will not have a next instruction; in this case the virus machine will reach a halting configuration, denoted by $i_t = \#$.

A *transition* or *computation step* of a super virus machine with OR channel parallelism V, from configuration C_t to configuration C_{t+1}, is performed by executing the instruction i_t as described above, and it is denoted as $C_t \Rightarrow_V C_{t+1}$ (or simply $C_t \Rightarrow C_{t+1}$ if V is clear from the context). A halting computation of a virus machine V is a finite sequence of configurations $\mathcal{C} = (C_0, C_1, \ldots, C_n)$, where C_0 is the initial configuration of V, the last configuration C_n is a halting configuration (that is, $i_n = \#$) and, for each $0 \leq t < n$, $C_t \Rightarrow_V C_{t+1}$. A non-halting computation consists of infinitely many successive configurations $\mathcal{C} = (C_i : i \in \mathbb{N})$.

4 An EXP-uniform solution to SAT

In this section, we provide an **EXP**-uniform family $\mathbf{V} = \{V(k) \mid k \in \mathbb{N}\}$ of super virus machines with OR channel parallelism, where each member of the family $V(\langle n, m\rangle)$ solves all the instances φ of SAT with n variables and m clauses.

Let φ be an instance of the SAT problem, that is, a CNF Boolean formula containing m clauses built over the set $Var(\varphi) = \{x_1, x_2, \ldots, x_n\}$ of variables. Let $s(\varphi) = \langle n, m\rangle$ be the size of this instance, and let $Cod(\varphi)$ be the following encoding, that produces the multiset to be given as input to the virus machine $V(\langle n, m\rangle)$ to specify that it should solve the instance φ of SAT:

- for all $1 \leq i \leq n$ and $1 \leq j \leq m$, $cod(\varphi)$ contains $\{v\}$ (that is, one virus) in the host with label (i, j, t), if the literal x_i occurs in clause C_j. Here, t denotes the logical value *true*;
- similarly, for all $1 \leq i \leq n$ and $1 \leq j \leq m$, $cod(\varphi)$ contains $\{v\}$ (that is, one virus) in the host with label (i, j, f), if the literal $\neg x_i$ occurs in clause C_j. Here, f denotes the logical value *false*.

We now define the super virus machine with OR channel parallelism

$$V(\langle n, m\rangle) = (\Gamma, H, I, D_H, D_I, G_C, n_1, \ldots, n_p, i_1, h_{out}),$$

where:

1. $\Gamma = \{v\}$,
2. $H = \{\langle i, j, t\rangle, \langle i, j, f\rangle, \langle i, j, aux\rangle \mid 1 \leq i \leq n, 1 \leq j \leq m\} \cup$
   $\{\langle j, k\rangle \mid 1 \leq j \leq m, k \in \{0, 1\}^n\} \cup$
   $\{k \mid k \in \{0, 1\}^n\} \cup \{yes\}$

3. $I = \{r \mid 1 \leq r \leq 2n+2\} \cup \{formula_j \mid 1 \leq j \leq m\} \cup \{assignments, output, end\}$
4. $D_H = (H \cup \{h_{out}\}, E_H, w_H), E_H = E_S \cup E_N$
   $E_S = \{(\langle i,j,t\rangle, \langle i,j,aux\rangle), (\langle i,j,aux\rangle, \langle i,j,t\rangle), (\langle i,j,f\rangle, \langle i,j,aux\rangle), (\langle i,j,aux\rangle, \langle i,j,f\rangle)$
   $E_N = E_N^1 \cup E_N^2,$
   $E_N^1 = E_N^{1,t} \cup E_N^{1,f},$
   $E_N^{1,t} = \{(\langle i,j,t\rangle, \langle j,k\rangle), (\langle i,j,aux\rangle, \langle j,k\rangle) \mid 1 \leq i \leq n, 1 \leq j \leq m, k \in \{0,1\}^n, k_i = 1\}$
   $E_N^{1,f} = \{(\langle i,j,f\rangle, \langle j,k\rangle), (\langle i,j,aux\rangle, \langle j,k\rangle) \mid 1 \leq i \leq n, 1 \leq j \leq m, k \in \{0,1\}^n, k_i = 0\}$
   $E_N^2 = (\langle j,k\rangle, k), (k, yes), (yes, env) \mid 1 \leq j \leq m, k \in \{0,1\}^n\}$
   $w(e) = 2$ for $e \in E_S$, $w(e) = 1$ for $e \in E_N$
5. $D_I = (I, E_I, w_I)$
   $E_I = \{(r, r+1) \mid 1 \leq r \leq 2n+1\} \cup \{(2n+2, assignments), (assignments, formula_1)\} \cup \{(formula_j, formula_{j+1}) \mid 1 \leq j \leq m-1\} \cup \{(formula_j, end) \mid 1 \leq j \leq m\} \cup \{(formula_m, output)\}$
   $w(e) = 2$ for $e \in \{(formula_j, formula_{j+1}) \mid 1 \leq j \leq m-1\} \cup \{(formula_m, output)\}$, $w(e) = 1$ otherwise
6. $G_C = (E_H \cup I, E_C)$
   $E_C = \{(2r+1, (\langle i,j,t\rangle, \langle i,j,aux\rangle)) \mid 1 \leq i \leq n, 1 \leq j \leq m, 0 \leq r \leq \lfloor \frac{n+1}{2} \rfloor\} \cup \{(2r, (\langle i,j,aux\rangle, \langle i,j,t\rangle)) \mid 1 \leq i \leq n, 1 \leq j \leq m, 1 \leq r \leq \lfloor \frac{n}{2} \rfloor\} \cup \{(n+1+2r+1, (\langle i,j,f\rangle, \langle i,j,aux\rangle)) \mid 1 \leq i \leq n, 1 \leq j \leq m, 0 \leq r \leq \lfloor \frac{n+1}{2} \rfloor\} \cup \{(n+1+2r, (\langle i,j,aux\rangle, \langle i,j,f\rangle)) \mid 1 \leq i \leq n, 1 \leq j \leq m, 1 \leq r \leq \lfloor \frac{n}{2} \rfloor\} \cup \{(n+1, e) \mid e \in E_N^{1,t}\} \cup \{(2n+2, e) \mid e \in E_N^{1,f}\} \{(assignments, (\langle j,k\rangle, k)) \mid 1 \leq j \leq m, k \in \{0,1\}^n\} \cup \{(formula_j, (k, yes)) \mid 1 \leq j \leq m, k \in \{0,1\}^n\} \cup \{(output, (yes, env))\}$
7. $n_i = 0$, for all $1 \leq i \leq q$
8. $h_{out} = env$ (the output is sent to the environment)

4.1 An overview of the computation

The computation of the virus machine $V(s(\varphi)) + cod(\varphi)$ is divided into four stages, that can be described as follows.

4.1.1 Generation stage

In this stage, 2^n viruses are going to be created in the origin hosts, where the input has been introduced. For this purpose, $2n+2$ computation steps are performed as follows. The first n steps generate 2^n viruses in the corresponding hosts $\langle i,j,t\rangle$, in such a way that 2^n viruses will be located either in those hosts or in $\langle i,j,aux\rangle$. This is obtained through the application of rules r, for $1 \leq r \leq n$. Next, instruction $n+1$ will send a virus to each host $\langle j,k\rangle$ in such a way that a virus will appear in such a host if and only if the corresponding truth assignment (denoted in the second part of the label k) makes true the clause C_j. In the same way, the next n steps are devoted to creating 2^n viruses either in hosts $\langle i,j,f\rangle$ or in hosts $\langle i,j,aux\rangle$. Since all the viruses from the previous task have been sent to other hosts, we are confident that they do not interfere with these steps. Instruction $2n+2$ works similarly to instruction $n+1$, but while the previous one sends a virus to those hosts whose truth assignment assigns 1 to variable x_i, in this case, it sends a virus to those hosts whose corresponding truth assignment assigns 0 to variable x_i. This stage takes $2n+2$ steps. The corresponding process is depicted in Figs. 1 and 2.

4.1.2 Assignments stage

In this stage, if a host $\langle j,k\rangle$ contains at least one virus, it means that the clause C_j is satisfied by the corresponding truth assignment encoded in k. Thus, the viruses are sent through the application of instruction *assignments*. This stage takes 1 computational step. For this, a single step is taken, represented in Fig. 3.

4.1.3 Formula checking stage

At this point, hosts k will have as many viruses as the number of clauses which are satisfied by the corresponding truth assignment. If there are exactly m viruses, it means that the corresponding truth assignment makes true exactly m clauses, hence it satisfies the whole formula φ. This case is obtained through the application of m consecutive instructions $formula_j$, for $1 \leq j \leq m$. The semantics of OR channel parallelism ensures that even if one host runs out of viruses, the other hosts can still send their viruses to the *yes* host. In this sense, if at least one host contains m viruses, then the instruction $formula_m$ will move the last virus to the host *yes*, going to the instruction *output*. Otherwise, at some point, an intermediate instruction $formula_j$ will not move any virus, bringing the computation to the instruction *end*. This stage takes at most m steps, and it is depicted in Fig. 4.

4.1.4 Output stage

If instruction $formula_m$ sent a virus to the host *yes*, then the instruction *output* will be activated, and will send a virus to the environment. On the other hand, if an instruction $formula_j$, for $1 \leq j \leq m$, does not move any virus, then there are no truth assignments that satisfy the original formula, and the computation halts without sending any virus to the environment. This process is performed by the module depicted in Fig. 5.

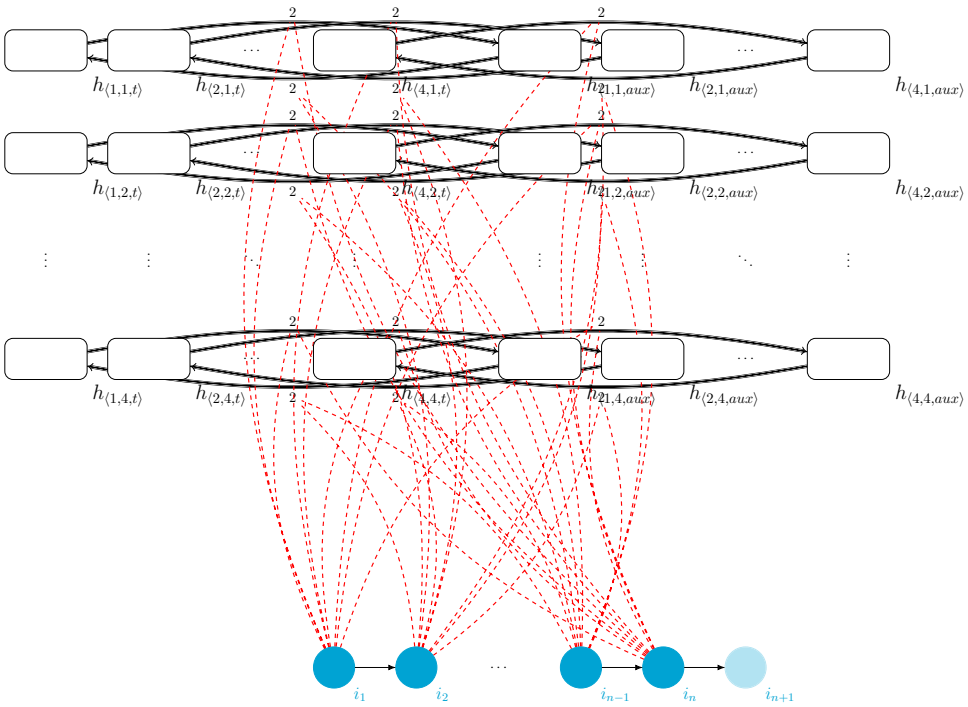

Fig. 1 Process for the generation of the viruses needed for the next stage

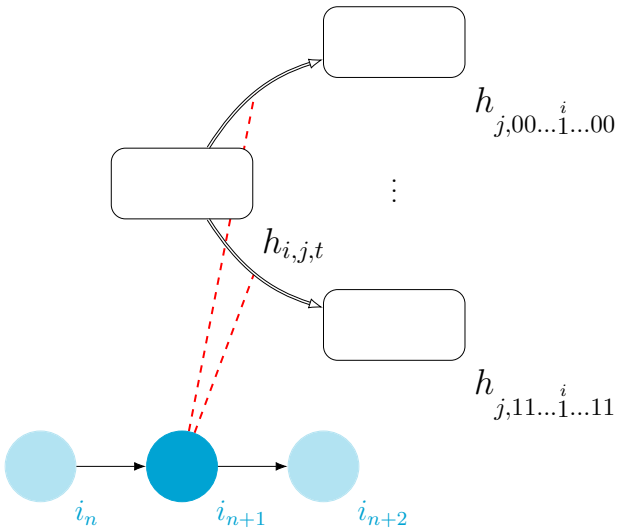

Fig. 2 The 2^n viruses from hosts $h_{i,j,r}$ are moved to the 2^n hosts $h_{j,k}$

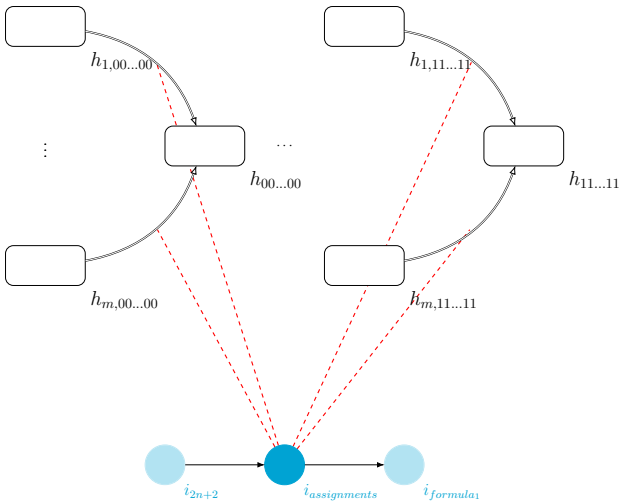

Fig. 3 All the viruses corresponding to the same truth assignment k are moved to the same host h_k

4.1.5 Computational resources

For this solution, the following computational resources are required:

- Number of initial hosts: $3nm + m2^n + 2^n + 1 \in O(m2^n)$.
- Number of initial channels: $m2^n + 2nm + 2^{n+1} + 2^n n + 1 \in O(m2^n + 2^{n+1} + n2^n)$.
- Maximum weight of channels: $2 \in O(1)$.
- Number of initial instructions: $2n + m + 5 \in O(n + m)$.
- Number of initial instruction connections: $2n + 2m + 3 \in O(n + m)$.
- Number of host-instruction control connections: $2mn^2 + m2^{n+1} + 2^n n + 2^n + 1 \in O(m2^{n+1})$.
- Number of initial viruses (apart from those given as input): $0 \in O(1)$.
- Maximum number of computation steps: $2n + m + 4 \in O(n + m)$.

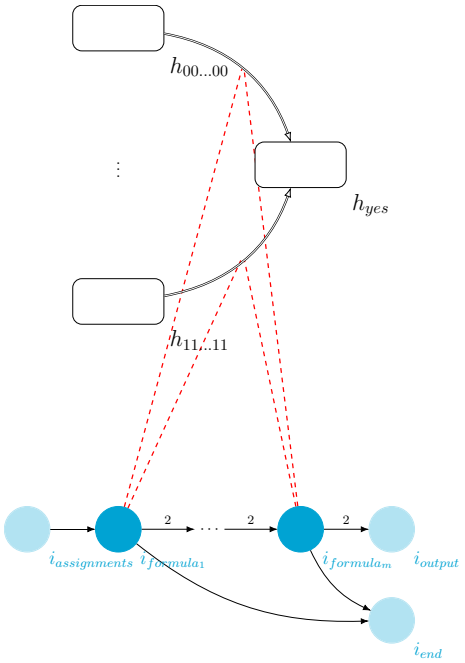

Fig. 4 If there exist m viruses in the host h_k, then the instruction i_{output} will be reached. Otherwise, the instruction i_{end} will be reached

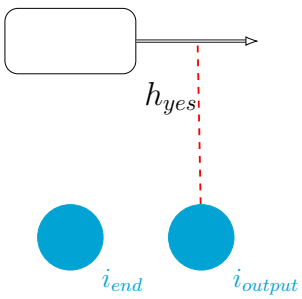

Fig. 5 If the instruction i_{output} is reached, then we know that there exists at least one truth assignment that makes the formula true, and therefore at least one virus is present in the host h_{yes}

Notice that the number of hosts, channels and host-instruction control connections is exponential with respect to the size of the input formula. Although we do not show it formally, it should be clear that the virus machine $V(s(\phi)) + cod(\phi)$, in its initial configuration, can be built by a deterministic Turing machine in exponential time with respect to n, the number of variables in the SAT instance to be solved. The precomputed resources contained in this virus machine then allow it to solve the specified SAT instance in linear time with respect to the number n of variables and the number m of clauses. This can be considered a time-space trade-off, much like it can be found in P systems with active membranes [25].

4.2 An example

To exemplify the computation process of the proposed super virus machine, let us consider the following instance of SAT: $\varphi \equiv (x_1 \vee x_2) \wedge (x_1 \vee \neg x_2)$. Since this formula has $n = 2$ variables and $m = 2$ clauses, the corresponding recognizer virus machine solving such an instance is $V(s(\varphi)) + cod(\varphi)$, that is, $V(\langle 2, 2\rangle) + cod((x_1 \vee x_2) \wedge (x_1 \vee \neg x_2))$. This recognizer virus machine is depicted in Fig. 6, where a virus is introduced as an input in hosts $h_{\langle 1,1,t\rangle}$, $h_{\langle 2,1,t\rangle}$, $h_{\langle 1,2,t\rangle}$ and $h_{\langle 2,2,f\rangle}$.

In the configuration C_6, exactly at the end of the generation stage, the number of viruses in the hosts $h_{j,k} (1 \leq j \leq 2, k \in \{0,1\}^2)$ are described in Table 1. These numbers match exactly the number of literals that make *true* the clause j with the corresponding truth assignment k.

At the end of the assignments stage, only hosts h_{10} and h_{11} will contain 2 viruses, that match exactly with the need of x_1 to take the value *true* for the formula to be satisfied. Since these two hosts have 2 viruses, instructions $i_{formula_1}$ and $i_{formula_2}$ will take the highest-weight path. Therefore, i_{output} will be selected and one virus will be sent to the environment, reaching a halting configuration in the next computation step.

5 Conclusions and future work

In this work, an efficient (linear time) **EXP**-uniform solution to the SAT problem has been presented by means of a family of super virus machines with OR channel parallelism. This solution exploits the classical schema of a brute-force algorithm, taking advantage of the OR channel parallelism present in this model. Taking into account that the provided solution needs an exponential number of hosts and channels from the beginning of the computation as precomputed resources, it would be interesting to look for ways to create such an exponential working space through other methods, such as the *mitosis* of the hosts, like it happens with division rules in membrane systems [36].

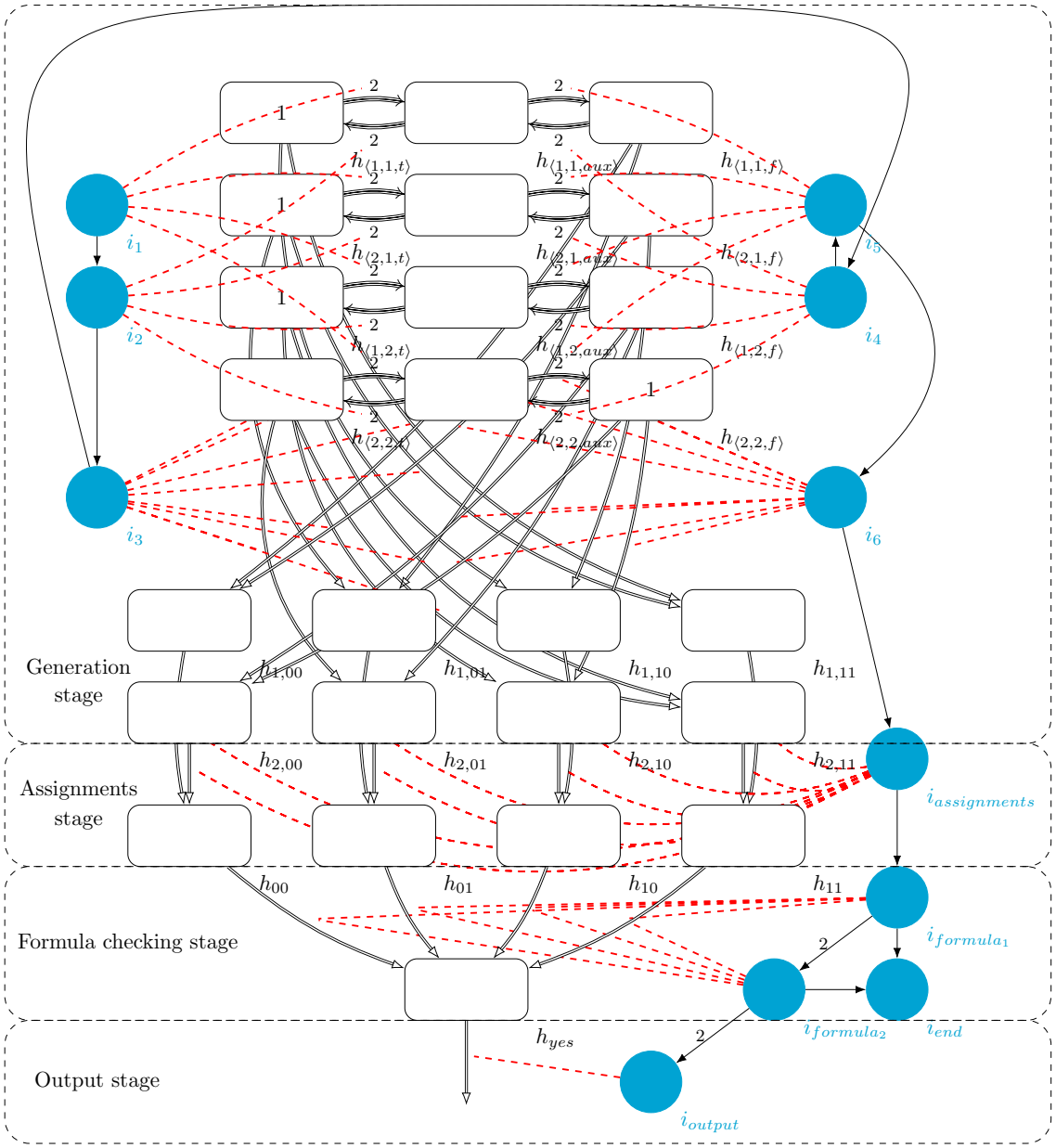

Fig. 6 The virus machine $V(\langle 2,2\rangle)$ with input $cod((x_1 \vee x_2) \wedge (x_1 \vee \neg x_2))$

Table 1 Number of viruses in hosts $h_{j,k}$ at configuration C_6

k	00	01	10	11
$j=1$	0	1	1	2
$j=2$	1	0	2	1

Acknowledgements D. Orellana-Martín acknowledges the support of the Zhejiang Lab BioBit Program (Grant no. 2022BCF05). The work of A. Leporati and C. Zandron was partially supported by the MUR under the grant "Dipartimenti di Eccellenza 2023-2027" of the Department of Informatics, Systems and Communication of the University of Milano-Bicocca, Italy.

Funding Funding for open access publishing: Universidad de Sevilla/CBUA.

References

1. Alhazov, A., Leporati, A., Manzoni, L., Mauri, G., & Zandron, C. (2021). Alternative space definitions for P systems with active membranes. *Journal of Membrane Computing, 3*(2), 87–96.
2. Ballesteros, K. J., Cailipan, D. P. P., de la Cruz, R. T. A., Cabarle, F. G. C., & Adorna, H. N. (2022). Matrix representation and simulation algorithm of numerical spiking neural P systems. *Journal of Membrane Computing, 4*(1), 41–55.
3. Baquero, F., Campos, M., Llorens, C., & Sempere, J. (2021). P systems in the time of COVID-19. *Journal of Membrane Computing, 3*(4), 246–257.
4. Battyányi, P., & Vaszil, G. (2020). Description of membrane systems with time Petri nets: Promoters/inhibitors, membrane dissolution, and priorities. *Journal of Membrane Computing, 2*(4), 341–354.
5. Ciencialová, L., Csuhaj-Varjú, E., Cienciala, L., & Sosík, P. (2019). P colonies. *Journal of Membrane Computing, 1*(3), 178–197.
6. de la Cruz, R. T. A., Cabarle, F. G. C., Macababayao, I. C. H., Adorna, H. N., & Zeng, X. (2021). Homogeneous spiking neural P systems with structural plasticity. *Journal of Membrane Computing, 3*(1), 10–21.
7. Díaz-Pernil, D., Gutiérrez-Naranjo, M. A., & Peng, H. (2019). Membrane computing and image processing: A short survey. *Journal of Membrane Computing, 1*(1), 58–73.
8. Dupaya, A. G. S., Galano, A. C. A. P., Cabarle, F. G. C., De La Cruz, R. T., Ballesteros, K. J., & Lazo, P. P. L. (2022). A web-based visual simulator for spiking neural P systems. *Journal of Membrane Computing, 4*(1), 21–40.
9. Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability. A guide to the theory on NP-completeness*. W.H. Freeman and Company.
10. Gheorghe, M., Lefticaru, R., Konur, S., Nicolescu, I., & Adorna, H. N. (2021). Spiking neural P systems: Matrix representation and formal verification. *Journal of Membrane Computing, 3*(2), 133–148.
11. Ionescu, M., Păun, G., & Yokomori, T. (2006). Spiking neural P systems. *Fundamenta Informaticae, 71*(2, 3), 279–308.
12. Langer, M., & Valenta, D. (2023). On evolving environment of 2D P colonies: Ant colony simulation. *Journal of Membrane Computing, 5*(3), 117–128.
13. Leporati, A., Manzoni, L., Mauri, G., Porreca, A. E., & Zandron, C. (2019). Characterizing PSPACE with shallow non-confluent P systems. *Journal of Membrane Computing, 1*(2), 75–84.
14. Leporati, A., Manzoni, L., Mauri, G., Porreca, A. E., & Zandron, C. (2020). Shallow laconic P systems can count. *Journal of Membrane Computing, 2*(4), 49–58.
15. Leporati, A., Manzoni, L., Mauri, G., Porreca, A. E., & Zandron, C. (2020). A Turing machine simulation by P systems without charges. *Journal of Membrane Computing, 2*(2), 71–79.
16. Leporati, A., Mauri, G., & Zandron, C. (2022). Spiking neural P systems: Main ideas and results. *Natural Computing, 21*(4), 629–649.
17. Liu, Y., Nicolescu, R., & Sun, J. (2020). Formal verification of cP systems using PAT3 and ProB. *Journal of Membrane Computing, 2*(2), 80–94.
18. Liu, Y., Nicolescu, R., & Sun, J. (2021). Formal verification of cP systems using Coq. *Journal of Membrane Computing, 3*(3), 205–220.
19. Lv, Z., Yang, Q., & Peng, H. (2021). Computational power of sequential spiking neural P systems with multiple channels. *Journal of Membrane Computing, 3*(4), 270–283.
20. Martín-Vide, C., Păun, G., Pazos, J., & Rodriguez-Paton, A. (2003). Tissue P systems. *Theoretical Computer Science, 296*(2), 295–326.
21. Nadizar, G., & Pietropolli, G. (2023). A grammatical evolution approach to the automatic inference of P systems. *Journal of Membrane Computing, 5*(3), 129–143.
22. Păun, G. (2000). Computing with membranes. *Journal of Computer and System Sciences, 61*(1), 108–143.
23. Păun, Gh. (2001). P systems with active membranes: Attacking NP-Complete problems. *Journal of Automata, Languages and Combinatorics, 6*(1), 75–90.
24. Pérez-Hurtado, I., Orellana-Martín, D., & Zhang, G. (2019). P-Lingua in two steps: Flexibility and efficiency. *Journal of Membrane Computing, 1*(2), 93–102.
25. Porreca, A., Leporati, A., Mauri, G., & Zandron, C. (2011). P systems with active membranes: Trading time for space. *Natural Computing, 10*(1), 167–182.
26. Qiu, C., Xue, J., Liu, X., et al. (2022). Deep dynamic spiking neural P systems with applications in organ segmentation. *Journal of Membrane Computing, 4*(4), 329–340.
27. Ramírez-de-Arellano, A., Orellana-Martín, D., & Pérez-Jiménez, M. J. (2023). Attacking cryptosystems by means of virus machines. *Nature Scientific Reports, 13*, 21831.
28. Ramírez-de-Arellano, A., Orellana-Martín, D., & Pérez-Jiménez, M. J. (2023). Generating, computing and recognizing with virus machines. *Theoretical Computer Science, 972*, 114077.
29. Ramírez-de-Arellano, A., Orellana-Martín, D., & Pérez-Jiménez, M. J. (2024). Bridges between spiking neural membrane systems and virus machines. *International Journal of Neural Systems, 34*(06), 2450034.
30. Ramírez-de-Arellano, A., Orellana-Martín, D., & Pérez-Jiménez, M. J. (2024). Parallel virus machines. *Journal of Membrane Computing, 6*, 211–221.
31. Ramírez-de-Arellano, A., Rodríguez-Gallego, J. A., Orellana-Martín, D., & Ivanov, S. (2023). Stochastic virus machines. In: Proceedings of the 19th Brainstorming Week on Membrane Computing, pp. 79–90. RGNC REPORT 1/2023, Sevilla, Spain.
32. Ramírez-de-Arellano, A., Valencia-Cabrera, L., Orellana-Martín, D., & Pérez-Jiménez, M. J. Super virus machines. *Intelligent Computing,* (submitted)
33. Romero-Jiménez, Á., Valencia-Cabrera, L., & Pérez-Jiménez, M. J. (2015). Generating diophantine sets by virus machines. In: Bio-Inspired Computing – Theories and Applications. BIC-TA 2015. Communications in Computer and Information Science, vol. 562, pp. 331–341. Springer, Berlin, Heidelberg.
34. Romero-Jiménez, Á., Valencia-Cabrera, L., Riscos-Núñez, A., & Pérez-Jiménez, M. J. (2015). Computing partial recursive functions by virus machines. *Membrane computing. CMC 2015. Lecture notes in computer science* (Vol. 9504, pp. 353–368). Springer.
35. Sempere, J. M. (2023). Modeling Markov sources and hidden Markov models by P systems. *Journal of Membrane Computing, 5*(3), 161–169.
36. Song, B., Li, K., Orellana-Martín, D., Pérez-Jiménez, M. J., & Pérez-Hurtado, I. (2022). A Survey of nature-inspired computing: Membrane computing. *ACM Computing Surveys, 54*(1), Article 22.
37. Sosík, P. (2019). P systems attacking hard problems beyond NP: A survey. *Journal of Membrane Computing, 1*(3), 198–208.

38. Sosík, P., Drastík, J., Smolka, V., et al. (2020). From P systems to morphogenetic systems: An overview and open problems. *Journal of Membrane Computing, 2*(4), 380–391.
39. Turlea, A., Gheorghe, M., Ipate, F., & Konur, S. (2019). Search-based testing in membrane computing. *Journal of Membrane Computing, 1*(4), 241–250.
40. Valencia-Cabrera, L., Pérez-Jiménez, M. J., Chen, X., Wang, B., & Zeng, X. (2015). Basic virus machines. In: 16th International Conference on Membrane Computing (CMC16), pp. 323–342.
41. Valencia-Cabrera, L., Perez-Hurtado, I., & Martinez-del Amor, M. A. (2020). Simulation challenges in membrane computing. *Journal of Membrane Computing, 2*(4), 1–11.
42. Verlan, S., Freund, R., Alhazov, A., et al. (2020). A formal framework for spiking neural P systems. *Journal of Membrane Computing, 2*(4), 355–368.
43. Wu, H., Ramírez-de-Arellano, A., Orellana-Martín, D., Wang, T., Wang, T., & Pérez-Jiménez, M. J. (2024). Channel parallel virus machine with production rules for power system fault diagnosis. *Journal of Membrane Computing* (online).
44. Yu, W., Wu, J., Chen, Y., et al. (2023). Fuzzy tissue-like P systems with promoters and their application in power coordinated control of microgrid. *Journal of Membrane Computing, 5*(1), 1–11.
45. Yu, W., Xiao, X., Wu, J., et al. (2023). Application of fuzzy spiking neural dP systems in energy coordinated control of multi-microgrid. *Journal of Membrane Computing, 5*(1), 69–80.
46. Zhao, S., Zhang, L., Liu, Z., et al. (2022). ConvSNP: A deep learning model embedded with SNP-like neurons. *Journal of Membrane Computing, 4*(1), 87–95.

David Orellana-Martín received his B.Sc. degree in 2014, his M.Sc. degree in 2016, and his Ph.D. degree in 2019 at the University of Seville. He received the Best Ph.D. thesis of 2019 Award from the International Membrane Computing Society. He was an ERCIM postdoctoral fellow at NTNU (Norway) from 2019 to 2021, a postdoctoral fellow at the University of Seville from 2021 to 2023, and currently is an Assistant Professor at the University of Seville. His main research interests are computational complexity theory, unconventional computing (especially membrane computing), machine learning, and high-performance computing. He is a member of the International Membrane Computing Society.

Claudio Zandron got the Ph.D., in computer science from the University of Milan in 2002. Since 2006, he is an Associate Professor at the Department of Informatics, Systems and Communication of the University of Milano-Bicocca, Italy. His research interests concern the areas of formal languages, molecular computing models, DNA computing, membrane computing, and computational complexity.

Alberto Leporati, PhD is an Associate Professor at the University of Milano-Bicocca, at the Department of Informatics, Systems and Communication. His research activity concerns the theory of computational complexity. In particular, he studies the computational power of models of computation which are inspired by the working of living cells (Membrane Computing) and the laws of quantum mechanics (Quantum Computing). On these topics, he has published more than 100 papers on international journals and in peer-reviewed proceedings of international conferences. He is also a member of the Steering Committee for the CMC and ACMC international conference series, and he serves as Vice President of the International Membrane Computing Society.