Solving NP-complete Problems by Spiking Neural P Systems with Budding Rules

Tseren-Onolt Ishdorj[1], Alberto Leporati[2], Linqiang Pan[3,4], Jun Wang[3]

[1] Computational Biomodelling Laboratory
Åbo Akademi University
Department of Information Technologies
20520 Turku, Finland
tishdorj@abo.fi

[2] Università degli Studi di Milano – Bicocca
Dipartimento di Informatica, Sistemistica e Comunicazione
Viale Sarca 336/14, 20126 Milano, Italy
alberto.leporati@unimib.it

[3] Key Laboratory of Image Processing and Intelligent Control
Department of Control Science and Engineering
Huazhong University of Science and Technology
Wuhan 430074, Hubei, People's Republic of China
junwangjf@gmail.com, lqpan@mail.hust.edu.cn

[4] Research Group on Natural Computing
Department of CS and AI, University of Sevilla
Avda Reina Mercedes s/n, 41012 Sevilla, Spain

Summary. Inspired by the growth of dendritic trees in biological neurons, we introduce spiking neural P systems with budding rules. By applying these rules in a maximally parallel way, a spiking neural P system can exponentially increase the size of its synapse graph in a polynomial number of computation steps. Such a possibility can be exploited to efficiently solve computationally difficult problems in deterministic polynomial time, as it is shown in this paper for the **NP**-complete decision problem SAT.

1 Introduction

Spiking neural P systems (SN P systems, for short) have been introduced in [5] as a new class of distributed and parallel computing devices, inspired by the neurophysiological behavior of neurons sending electrical impulses (*spikes*) along axons to other neurons. SN P systems can also be viewed as an evolution of P systems [19, 16] corresponding to a shift from *cell-like* to *neural-like* architectures. We recall that this biological background has already led to several models in the area of neural computation, e.g., see [13, 14, 4].

In SN P systems the cells (also called *neurons*) are placed in the nodes of a directed graph, called the *synapse graph*. The contents of each neuron consist of a number of copies of a single object type, called the *spike*. Every cell may also contain a number of *firing* and *forgetting* rules. Firing rules allow a neuron to send information to other neurons in the form of electrical impulses (also called spikes) which are accumulated at the target cells. The applicability of each rule is determined by checking the contents of the neuron against a regular set associated with the rule. In each time unit, if a neuron can use some of its rules then one of such rules must be used. The rule to be applied is nondeterministically chosen. Thus, the rules are used in a sequential manner in each neuron, but neurons function in parallel with each other. Observe that, as usually happens in membrane computing, a global clock is assumed, marking the time for the whole system, hence the functioning of the system is synchronized. When a cell sends out spikes it becomes "closed" (inactive) for a specified period of time, that reflects the refractory period of biological neurons. During this period, the neuron does not accept new inputs and cannot "fire" (that is, emit spikes). Another important feature of biological neurons is that the length of the axon may cause a time delay before a spike reaches its target. In SN P systems this delay is modeled by associating a delay parameter to each rule which occurs in the system. If no firing rule can be applied in a neuron, there may be the possibility to apply a *forgetting rule*, that removes from the neuron a predefined number of spikes.

The computational efficiency of SN P systems has been recently investigated in a series of works [2, 6, 9, 11, 10]. In [12] it has been proved that a deterministic SN P system of polynomial size cannot solve an **NP**-complete problem in a polynomial time, unless **P=NP**. Hence, under the assumption that $\mathbf{P} \neq \mathbf{NP}$, efficient solutions to **NP**-complete problems cannot be obtained without introducing features which enhance the efficiency, such as pre-computed resources, ways to exponentially grow the workspace during the computation, nondeterminism, and so on. Indeed, in the framework of SN P systems, most of the solutions to computationally hard problems exploit the power of nondeterminism [11, 10, 12] or use pre-computed resources of exponential size [2, 6, 9, 7].

The possibility of using SN P systems to solve computationally hard problems by using some (possibly exponentially large) pre-computed resources has been first presented in [6], that contains a description of a uniform family of SN P systems with pre-computed resources of exponential size that solves all the instances of the **NP**-complete decision problem SAT in a polynomial time. In the present paper we complement the study exposed in [6], by describing an SN P system that first builds the necessary resources (by exponentially increasing its workspace in a polynomial time), and then uses such resources to solve the SAT problem. To this purpose, we extend the SN P systems given in [6] by introducing *neuron budding rules*. We show that SN P systems with budding rules can grow an exponential size synapse graph in a time which is polynomial with respect to the size of the instances of the problem we want to solve. Then, the systems themselves can be used to solve such instances. All the systems we will propose work in a *deterministic* way.

The biological motivation for the mechanism that we use to expand the synapse graph of SN P systems comes from the growth of dendritic trees in biological neurons [20]. It is known that the human brain is made up of about 100 billion cells. Almost all brain cells are formed before birth. Dendrites (from the Greek, "tree") are the branched projections of a neuron. The point at which the dendrites of a cell come into contact with the dendrites of another cell is where the miracle of information transfer (communication) occurs. *Brain cells can grow as many as one billion of dendrite connections* – a universe of touch points. The greater the number of dendrites, the more information can be processed. Dendrites grow as a result of stimulation from and interaction with their environment. With limited stimulation there is limited growth; with no stimulation, dendrites actually retreat and disappear. The microscope photographs illustrated in Figure 1 show actual dendrite development. Dendrites begin to emerge from a single neuron (brain cell) and develop into a cluster of touch points seeking to connect with dendrites from other cells.

In the framework of SN P systems, the dendrite connection points are modelled as abstract neurons, while the branches of dendrite trees are modelled as abstract synapses. A new connection between dendrites coming from two different neuron cells is understood as a newly created synapse. In this way, new neurons and new synapses can be produced during the growth of a dendrite tree. The formal definition of neuron budding rule and its semantics will be given in Section 2.

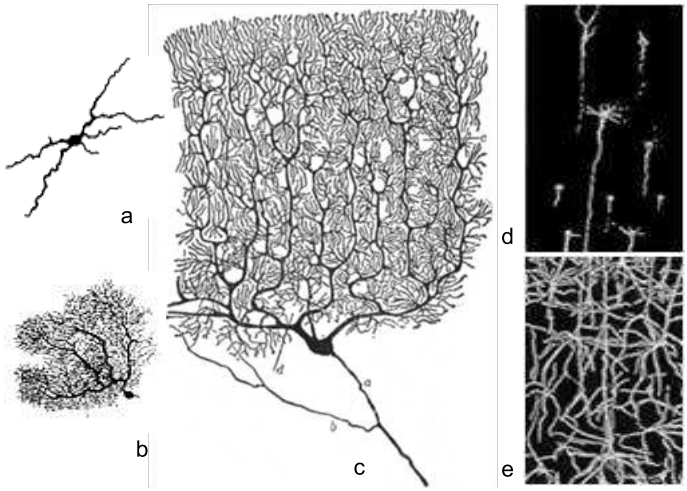

Fig. 1. A growing neuron: a. dendrites begin to emerge from a single neuron, b. dendrites developed into a cluster of touch points; c. Ramon y Cajal, Santiago. Classical drawing: Purkinje cell; d. newborn neuron dendrites, e. three months later. Photos from Tag Toys [20]

2 SN P systems with budding rules

A *spiking neural P system with budding rules*, of initial degree $m \geq 1$, is a construct of the form

$$\Pi = (O, \Sigma, H, syn, R, in, out),$$

where:

1. $O = \{a\}$ is the singleton alphabet (a is called *spike*);
2. $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_m\}$ is a finite set of initial neurons;
3. H is a finite set of *labels* for neurons;
4. $syn \subseteq H \times H$ is a finite set of *synapses*, with $(i, i) \notin syn$ for $i \in H$;
5. R is a finite set of *developmental rules*, of the following forms:
   (1) *neuron budding rule* $x[\ ]_i \rightarrow y[\ ]_j$, where $x \in \{(k, i), (i, k), \lambda\}$, $y \in \{(i, j), (j, i), \lambda\}$, $i, j, k \in H$, $i \neq k$, $i \neq j$;
   (2) *extended firing* (also called *spiking*) rule $[E/a^c \rightarrow a^p; d]_i$, where $i \in H$, E is a regular expression over a, and $c \geq 1$, $p \geq 0$, $d \geq 0$ are integer numbers, with the restriction $c \geq p$.
6. $in, out \in H$ indicate the *input* and the *output* neurons of Π.

Note that the definition of SN P systems with budding rules is slightly different from the usual definition of SN P systems given in the literature, where the neurons that occur in the system are explicitly listed as $\sigma_i = (n_i, R_i)$, $1 \leq i \leq m$, where R_i is the set of rules associated with neuron σ_i, and n_i is the number of spikes it contains in the initial configuration of the system. First of all, only the *structure* of the system is given in our definition; the presence of spikes (if any) in the initial configuration is specified at the beginning of each computation. Further, i is considered as the label of neuron σ_i. In SN P systems with budding rules it is possible to create new neurons in the course of a computation; hence the system may contain, in a given configuration, several neurons that are labelled with the same element of H. Nonetheless, with a slight abuse of notation in what follows we will refer to any neuron having the label $i \in H$ by calling it σ_i.

Considering the budding rule $x[\ ]_i \rightarrow y[\ ]_j$, its left hand side describes the neuron σ_i with a synapse x connected with one of its neighbouring neurons, to which the rule is supposed to be applied. The right hand side describes the result of the rule application, that is, the newly created neuron σ_j and synapse y. Note that for the sake of simplicity, in the rule notation we omit to repeat the contents of the left hand side of the rule in the right hand side. We say that the rule is *restricted* because only one neighbouring neuron is considered in each side of the rule.

A budding rule can be applied only if the neighbourhood of the associated neuron is exactly as described in the left hand side of the rule, in other words, $x = X$ where X is the current set of synapses of neuron σ_i. As a result of the rule application, a new neuron σ_j and a synapse y are established, provided that they do not already exist; if a neuron with label j already exists in the system but no synapse of type y exists, then only the synaptic connection y between the neurons

σ_i and σ_j is established; no new neuron with label j is budded. We stress here that the application of budding rules does not depend on the spikes contained into the neuron. Budding rules are applied in a maximally parallel way: if the neighbourhood of neuron σ_i enables several budding rules, then all these rules are applied in parallel; as a result, several new neurons and synapses are produced (which corresponds to have several branches at a touch point in the dendrite tree). Note that the way of using neuron budding rules is different with respect to the usual way in which P systems with active membranes use cell division or cell creation rules, where at most one of these rules can be applied inside each membrane during a computation step.

Extended firing rules are defined as usually done in SN P systems. If an extended firing rule $[E/a^c \to a^p; d]_i$ has $E = a^c$, then we will write it in the simplified form $[a^c \to a^p; d]_i$; similarly, if a rule $[E/a^c \to a^p; d]_i$ has $d = 0$, then we can simply write it as $[E/a^c \to a^p]_i$; hence, if a rule $[E/a^c \to a^p; d]_i$ has $E = a^c$ and $d = 0$, then we can write $[a^c \to a^p]_i$. A rule $[E/a^c \to a^p]_i$ with $p = 0$ is written in the form $[E/a^c \to \lambda]_i$ and is called an *extended forgetting* rule. Rules of the types $[E/a^c \to a; d]_i$ and $[a^c \to \lambda]_i$ are said to be *standard*. However, even in this case we do not require that if a forgetting rule is enabled then no firing rules are also enabled at the same time in the same neuron, as it happens in standard SN P systems.

If a neuron σ_i contains k spikes and $a^k \in L(E), k \geq c$, then the rule $[E/a^c \to a^p; d]_i$ is enabled and can be applied. This means consuming (removing) c spikes (thus only $k - c$ spikes remain in neuron σ_i); the neuron is fired, and it produces p spikes after d time units. If $d = 0$, then the spikes are emitted immediately; if $d = 1$, then the spikes are emitted in the next step, etc. If the rule is used in step t and $d \geq 1$, then in steps $t, t + 1, t + 2, \ldots, t + d - 1$ the neuron is closed (this corresponds to the refractory period from neurobiology), so that it cannot receive new spikes (if a neuron has a synapse to a closed neuron and tries to send a spike along it, then that particular spike is lost). In the step $t + d$, the neuron spikes and becomes open again, so that it can receive spikes (which can be used starting with the step $t + d + 1$, when the neuron can again apply rules). Once emitted from neuron σ_i, the p spikes reach immediately all neurons σ_j such that there is a synapse going from σ_i to σ_j and which are open, that is, the p spikes are replicated and each target neuron receives p spikes; as stated above, spikes sent to a closed neuron are "lost", that is, they are removed from the system. In the case of the output neuron, p spikes are also sent to the environment. Of course, if neuron σ_i has no synapse leaving from it, then the produced spikes are lost. If the rule is a forgetting one of the form $[E/a^c \to \lambda]_i$, then, when it is applied, $c \geq 1$ spikes are removed. When a neuron is closed, none of its rules can be used until it becomes open again.

In each time unit, if a neuron σ_i can use one of its rules, then a rule from R *must* be used. If the neighbourhood of neuron σ_i enables several budding rules, then all these rules are applied in parallel. If several spiking rules are enabled in neuron σ_i, then only one of them is nondeterministically chosen. If both spiking

rules and budding rules are enabled in the same computation step, then one type of rules is nondeterministically chosen. When a neuron budding rule is applied, at this step the associated neuron is closed, and thus it cannot receive spikes. In the next step, the neurons obtained by budding will be open.

The *configuration* of the system is described by its topology structure, the number of spikes associated with each neuron, and the *state* of each neuron (open or closed). We emphasize that the system introduced here contains no spikes in the initial configuration. Using the rules as described above, one can define *transitions* among configurations. Any sequence of transitions starting in the initial configuration is called a *computation*. A computation *halts* if it reaches a configuration where all the neurons are open and no rule can be used.

In what follows, we give an example to make the application of budding rules transparent. Neither spiking nor forgetting rules are used.

An example. Let Π_1 be an SN P system with budding rules, whose initial topological structure (composed by a single neuron σ_1) is shown in the left hand side of Figure 2. Let Π_1 contain the following six budding rules:

a. $\lambda[\ ]_1 \rightarrow (1,2)[\ ]_2$,
b. $(1,2)[\ ]_2 \rightarrow (3,2)[\ ]_3$,
c. $(1,2)[\ ]_2 \rightarrow (2,4)[\ ]_4$,
d. $(2,3)[\ ]_3 \rightarrow (3,5)[\ ]_5$,
e. $(2,4)[\ ]_4 \rightarrow (4,6)[\ ]_6$,
f. $(4,6)[\ ]_6 \rightarrow (6,3)[\ ]_3$.

In the initial configuration, neuron σ_1 has no neighbourhood and only rule *a*. is enabled. The application of rule *a*. produces a new neuron σ_2 with a synapse (1,2) connecting it with σ_1. Now both neurons σ_1 and σ_2 have a neighbourhood (each one being the neighbourhood of the other), since a synaptic connection exists between them. In this circumstance, rule *a*. is disabled while rules *b*. and *c*. are enabled and may be applied in parallel to neuron σ_2. When these two rules are applied two new neurons σ_3 and σ_4 are created, with the associated synapses (3,2) and (2,4). In the resulting configuration, rules *b*. and *c*. are disabled since now neuron σ_2 has three neighbours. At this step only rule *e*. can be applied to neuron σ_4, producing a new neuron σ_6 with a synaptic connection (4,6). Note that at this step rule *d*. was not enabled as the synapse of neuron σ_3 is (3,2), instead of (2,3) as required by the rule. Now only rule *f*. is enabled, which creates only the synapse (6,3) because neuron σ_3 already exists. From now on no rule is enabled, and thus the computation halts.

3 SN P systems solving SAT

Let us now consider the **NP**-complete decision problem SAT [8, p. 39]. The instances of SAT depend upon two parameters: the number n of variables, and the

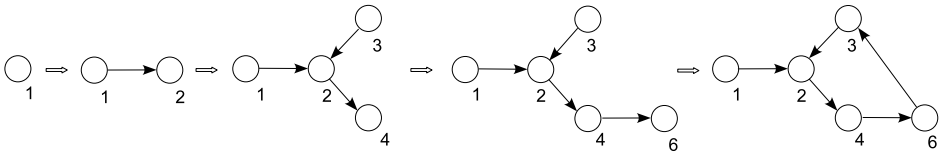

Fig. 2. Evolution of the structure of the SN P system Π_1, as the effect of the application of budding rules

number m of clauses. We recall that a *clause* is a disjunction of literals, occurrences of x_i or $\neg x_i$, built on a given set $X = \{x_1, x_2, \ldots, x_n\}$ of Boolean variables. Without loss of generality, we can avoid the clauses in which the same literal is repeated or both the literals x_i and $\neg x_i$, for any $1 \leq i \leq n$, occur. In this way, a clause can be seen as a *set* of at most n literals. An *assignment* of the variables $x_1, x_2, \ldots, x_n$ is a mapping $a : X \to \{0, 1\}$ that associates to each variable a truth value. The number of all possible assignments to the variables of X is 2^n. We say that an assignment *satisfies* the clause C if, assigned the truth values to all the variables which occur in C, the evaluation of C (considered as a Boolean formula) gives 1 (*true*) as a result.

We can now formally state the SAT problem as follows.

Problem 1. NAME: SAT.

- INSTANCE: a set $C = \{C_1, C_2, \ldots, C_m\}$ of clauses, built on a finite set $\{x_1, x_2, \ldots, x_n\}$ of Boolean variables.
- QUESTION: is there an assignment to the variables $x_1, x_2, \ldots, x_n$ that satisfies all the clauses in C?

Equivalently, we can say that an instance of SAT is a propositional formula $\gamma_n = C_1 \wedge C_2 \wedge \cdots \wedge C_m$, expressed in the conjunctive normal form as a conjunction of m clauses, where each clause is a disjunction of literals built using the Boolean variables $x_1, x_2, \ldots, x_n$. With a little abuse of notation, from now on we will denote by SAT(n, m) the set of instances of SAT which have n variables and m clauses.

In [6], a uniform family $\{\Pi_{SAT}(\langle n, m\rangle)\}_{n,m\in\mathbf{N}}$ of SN P systems was built such that for all $n, m \in \mathbb{N}$ the system $\Pi_{SAT}(\langle n, m\rangle)$ solves all the instances of SAT(n, m) in a number of steps which is quadratic in n and linear in m. Here $\langle n, m\rangle$ denotes the natural number obtained by applying the Cantor bijection to the pair (n, m) of natural numbers; so doing, the family of P systems depends upon one parameter instead of two. We assume that the reader is familiar with the construction given in [6]; for his convenience, here we summarize the structure and functioning of the system $\Pi_{SAT}(\langle n, m\rangle)$. In the next section, we are going to build such a system by means of budding rules.

Because the construction is uniform, we need a way to encode any given instance γ_n of SAT(n, m). As stated above, each clause C_i of γ_n can be seen as a disjunction of at most n literals, and thus for each $j \in \{1, 2, \ldots, m\}$ either x_j

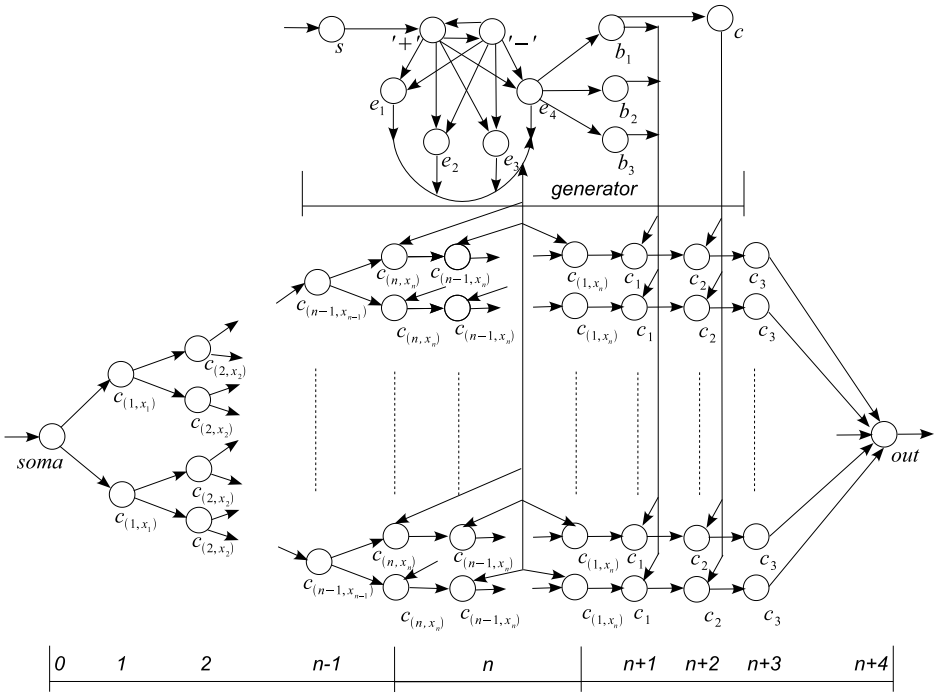

Fig. 3. A SN P system structure devoted to solve all the instances of SAT(n, m)

occurs in C_i, or $\neg x_j$ occurs, or none of them occurs. In order to distinguish these three situations we define the *spike variables* α_{ij}, for $1 \leq i \leq m$ and $1 \leq j \leq n$, as variables whose values are amounts of spikes, and we assign to them the following values:

$$\alpha_{ij} = \begin{cases} a & \text{if } x_j \text{ occurs in } C_i \\ a^2 & \text{if } \neg x_j \text{ occurs in } C_i \\ \lambda & \text{otherwise} \end{cases} \tag{1}$$

So doing, clause C_i will be represented by the sequence $\alpha_{i1}\alpha_{i2}\cdots\alpha_{in}$ of spike variables; in order to represent the entire formula γ_n we just concatenate the representations of the single clauses, thus obtaining the sequence $\alpha_{11}\alpha_{12}\cdots\alpha_{1n}\alpha_{21}\alpha_{22}\cdots\alpha_{2n}\cdots\alpha_{m1}\alpha_{m2}\cdots\alpha_{mn}$. As an example, the representation of $\gamma_3 = (x_1 \vee \neg x_2) \wedge (x_1 \vee x_3)$ is the sequence $aa^2\lambda a\lambda a$.

The system structure is composed of $n + 5$ layers, as illustrated in Figure 3. The first layer (numbered by 0) is used to insert into the system the representation of the instance of SAT(n, m) to be solved, encoded as stated above. Note that each layer from 1 to n contains two times the neurons contained in the previous layer. In this way we obtain in the n-th layer 2^n copies of a subsystem which is a sequence of n neurons; each subsystem is bijectively associated to one of the

possible assignments to the variables $x_1, x_2, \ldots, x_n$. The neurons that occur in each subsystem are of two types: f and t. The type of a neuron indicates that the corresponding Boolean variable is assigned with the Boolean value $t(rue)$ or $f(alse)$, respectively. These subsystems, together with the so called *generator*, have a very specific function in the overall SN P system: to test (in parallel) all possible assignments against a given clause.

The assignment is performed by sending 3 spikes to all the neurons labelled with t, and 4 spikes to all the neurons labelled with f. This means that neurons e in the generator will have three synapses going to neurons t and four synapses towards neurons f. All these spikes arrive every n computation steps, when the spikes indicated by the spike variables α_{ij} that correspond to a clause of γ_n are contained into the subsystems of layer n. This process is started by putting one spike in neuron s at the beginning of the computation. The delay associated with the rule contained in neuron s allows to send the first spikes from neurons e to neurons t and f exactly when the first clause is contained in layer n.

Recall our encoding of literals in the clauses (1): we have 0 spikes if the variable does not occur in the clause, 1 spike if it occurs non negated, and 2 spikes if it occurs negated. These spikes are added with those representing the assignments,

	Assign. to x_j	*Literal*	*N. of spikes*	*Truth value of* C_i
Neuron t	*true*	$x_j \notin C_i$	$3+0=3$	*?*
	true	$x_j \in C_i$	$3+1=4$	*true*
	true	$\neg x_j \in C_i$	$3+2=5$	*?*
Neuron f	*false*	$x_j \notin C_i$	$4+0=4$	*?*
	false	$x_j \in C_i$	$4+1=5$	*?*
	false	$\neg x_j \in C_i$	$4+2=6$	*true*

Table 1. Number of spikes resulting from the assignment in the neurons of layer n, and its effect on the truth value of the clause

and the possible results are illustrated in Table 1. From this table we can see that if a neuron labelled with t receives a total number of 4 spikes then the corresponding variable occurs non negated in the clause and is assigned the truth value *true*; we can immediately conclude that the clause is satisfied, and thus the neuron sends one spike towards the next layer. Similarly, if a neuron labelled with f receives 6 spikes then the corresponding variable occurs negated in the clause and is assigned the truth value *false*; also in this case we can immediately conclude that the clause is satisfied, and the neuron signals this event by sending one spike towards the next layer. In all the other cases we cannot conclude anything on the truth value of the clause, and thus no spike is emitted.

All the spikes which are emitted by neurons t and f are propagated through the neurons that compose layer n, until they reach the corresponding neuron σ_1 in layer $n+1$. Such a neuron is designed to make neuron σ_2 (in layer $n+2$) retain only one spike from those received by layer n. Hence, those assignments that satisfy the clause produce a single spike in the corresponding neuron σ_2; such a spike is accumulated in the associated neuron σ_3 (in layer $n+3$), that operates like a counter. When the first clause of γ_n has been processed, the second enters into the system (in n steps) and takes place in the subsystems; then all possible assignments are tested against this clause, and so on for all the clauses. When all the m clauses of γ_n have been processed, neurons σ_3 in layer $n+3$ contain each the number of clauses which are satisfied by the corresponding assignment. The neurons that contain m spikes fire, sending one spike to neuron σ_{out}, thus signalling that their corresponding assignment satisfies all the clauses of the instance. Neuron σ_{out} operates like an OR gate: it fires if and only if it contains at least one spike, that is, if and only if at least one of the assignments satisfies all the clauses of γ_n. Further technical details will be presented in the last part of the next section.

4 A uniform solution to SAT by SN P systems with budding rules

In this section we show that the pre-computed structures which are used in [6] to solve the instances of $\text{SAT}(n,m)$ can be built in a polynomial time by SN P systems with budding rules. The SN P system with budding rules that we are going to define is composed of two subsystems: a first subsystem builds the structure of a second subsystem, that solves the instances of $\text{SAT}(n,m)$ as described in the previous section. For the sake of simplicity, we avoid to use the neuron budding and the spiking rules at the same time in each subsystem.

Formally, the SN P system with budding rules is defined as

$$\Pi = (O, \Sigma, H, syn, R, soma, out)$$

where:

1. $O = \{a\}$ is the singleton alphabet;
2. $\Sigma = \{\sigma_i \mid i \in H_0\}$ is the set of initial neurons;
3. H is a finite set of labels for neurons, and
   $H \supseteq H_0 = \{soma, out, e_0, e_1, e_2, e_3, b_1, b_2, b_3, c, s, +, -\}$ is the set of labels for the neurons initially given;
4. $syn \subseteq H \times H$ is a finite set of synapses, with $(i,i) \notin syn$ for $i \in H$), and
   $syn \supseteq syn_0 = \{(e, e_i) \mid 0 \leq i \leq 3, e \in \{+,-\}\} \cup \{(e_0, b_i) \mid 1 \leq i \leq 3\} \cup \{(b_3, c), (s,+), (+,-), (-,+), \lambda\}$ is the set of synapses initially in use;
5. *soma* and *out* are the labels for the *input* and *output* neuron, respectively;
6. R is a set of *neuron budding* and *extended spiking* rules defined as follows.

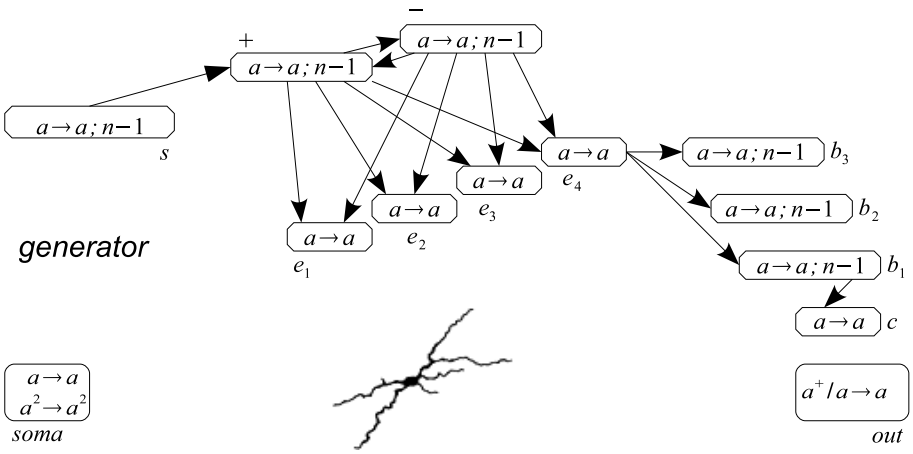

Fig. 4. The initial topological structure (newly born dendrite) of the SN P system Π: the input (*soma*) and the output (*out*) neurons, and the *generator*

Building the system structure.

The system initially contains an input neuron σ_{soma}, an output neuron σ_{out}, and a sub-structure G (named the *generator*) which is composed of the set of neurons specified in Σ and the set of synapses from syn_0, arranged as illustrated in Figure 4.

The generator is governed only by neuron budding rules, and is controlled by the labels of budding neurons and by the synapses created during the computation. The system construction algorithm consists of two phases:

A. Generation of the *dendritic-tree* sub-structure (the layers from 0 to n in Figure 3) and assignment of the truth values to the n Boolean variables. The process starts from the initial neuron σ_{soma} (the root node) and produces 2^n neurons in n steps. The label of each neuron in layer n encodes an associated truth assignment.
B. Completion of the network structure. The neurons in the n-th layer of the system establish connections with the *generator*, according to the truth assignments represented in those neurons. The structure is then further expanded by three layers, and finally all the neurons in the last layer are connected with the output neuron σ_{out}.

Let us now describe in depth each of these phases.

Phase A. In this phase of computation, the dendritic tree (which is a complete binary tree) is generated in n steps by applying budding rules of type $\mathbf{a_1}$), described below, starting from an initial neuron σ_{soma}. The dendritic tree generation process is controlled by the labels of the neurons as well as by the synapses generated so far.

It is worth to note that, since the truth assignments associated with the neurons in n-th layer are encoded in the labels of those neurons, also the truth assignments to the variables $x_1, x_2, \ldots, x_n$ are generated during the construction of the dendritic tree.

The label of a neuron σ_c in layer i is a sequence of the form

$$c = (i, x_i^{(p)}) = (i, x_i(1) = p) = (i, p, x_{k2}, \ldots, x_{ii}),$$

with $p \in \{t, f\}$, where the first entry (i) indicates the number of layer, while $x_i^{(p)}$ is a subsequence of length i formed by the Boolean values t and f that have been generated up to now, that represents a truth assignment to the variables $x_1, x_2, \ldots, x_i$. The component p in $x_i^{(p)}$ indicates that the first entry of the subsequence is exactly p.

An almost complete structure of the SN P system that solves the instances of SAT$(2, m)$ is illustrated in Figure 5. It is worth to follow its construction.

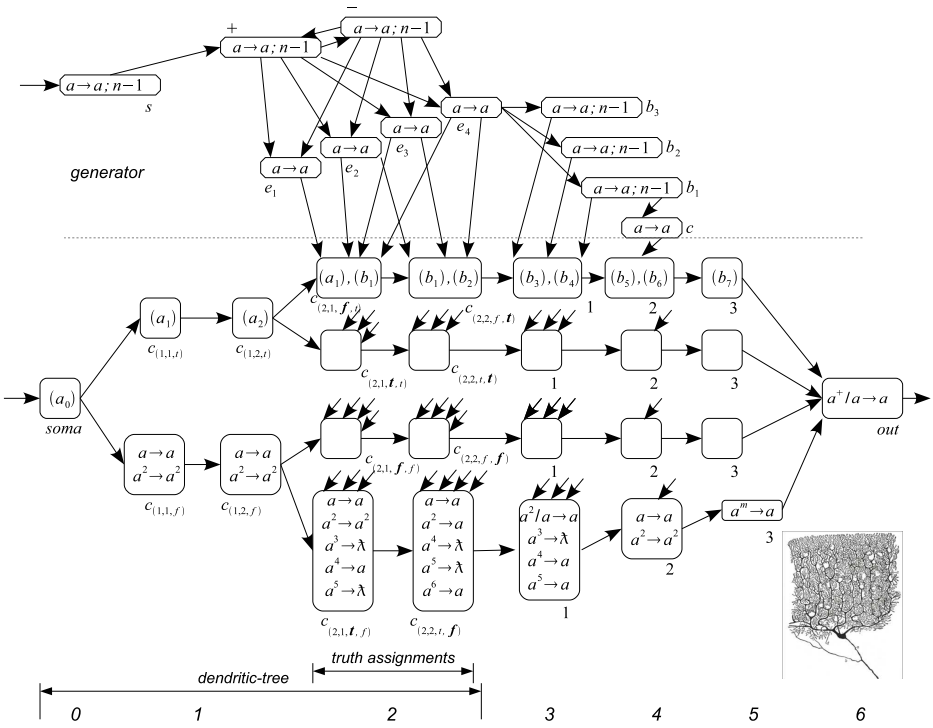

Fig. 5. An almost complete structure (maturated dendrite tree) of the P system for solving the instances of SAT$(2, m)$. The neuron budding rules used in each computation step are indicated by their labels in the corresponding neurons. Some of the spiking rules are also indicated

a₁) $(c_{(i,x_{i-1})}, c_{(i,x_i)})[\ ]_{c_{(i,x_i)}} \to (c_{(i,x_i)}, c_{(i+1,p,x_i)})[\ ]_{c_{(i+1,p,x_i)}}$,
$0 \le i \le n-1,\ p \in \{t,f\},\ x_i \in \{t,f\}^i$, $(c_{(-1,x_{-1})}, c_{(0,x_0)}) = \lambda$, $c_{(0,\lambda)} = soma$.
The computation starts by applying two rules of type **a₁)**, for $i = 0$, to the input neuron $\sigma_{c_{soma}}$. These two rules are:

$$[\ ]_{c_{soma}} \to (c_{soma}, c_{(1,t)})[\ ]_{c_{(1,t)}}, \quad \text{and} \quad [\ ]_{c_{soma}} \to (c_{soma}, c_{(1,f)})[\ ]_{c_{(1,f)}},$$

where $(c_{soma}, c_{(1,t)}), (c_{soma}, c_{(1,f)}) \in syn$.
The left hand side of each rule (where $\lambda \in syn_0$ is omitted) requires that its interaction environment be empty, i.e., no synapse exists connected to neuron $\sigma_{c_{soma}}$. As the left hand sides of both these rules are the same, and satisfy the constraints posed on the interaction environment of neuron $\sigma_{c_{soma}}$, they are applied simultaneously. As a result, two new neurons are budded: $\sigma_{c_{(1,t)}}$, with a synapse $(c_{soma}, c_{(1,t)})$ coming from the father neuron, and $\sigma_{c_{(1,f)}}$, connected with the father neuron by a synapse $(c_{soma}, c_{(1,f)})$. The symbols t and f in the neuron labels indicate the truth values *true* and *false*, respectively, and can be regarded as the two truth assignments (t) and (f) of length 1 for a single Boolean variable x_1. The first layer of the dendritic tree is thus established, and rules of type **a₁)** cannot be applied anymore, since the interaction environment of neuron $\sigma_{c_{soma}}$ has changed.
At the second computation step ($i = 1$), the following two rules are enabled and can be applied to each of the newly created neurons:

$$(c_{soma}, c_{(1,t)})[\ ]_{c_{(1,t)}} \to (c_{(1,t)}, c_{(2,f,t)})[\ ]_{c_{(2,f,t)}},$$
$$(c_{soma}, c_{(1,t)})[\ ]_{c_{(1,t)}} \to (c_{(1,t)}, c_{(2,t,t)})[\ ]_{c_{(2,t,t)}}$$

for $\sigma_{c_{(1,t)}}$, and

$$(c_{soma}, c_{(1,f)})[\ ]_{c_{(1,f)}} \to (c_{(1,f)}, c_{(2,t,f)})[\ ]_{c_{(2,t,f)}},$$
$$(c_{soma}, c_{(1,f)})[\ ]_{c_{(1,f)}} \to (c_{(1,f)}, c_{(2,f,f)})[\ ]_{c_{(2,f,f)}}$$

for $\sigma_{c_{(1,f)}}$. The former pair of rules yields to two new neurons having label $c_{(2,f,t)}$ and $c_{(2,t,t)}$, respectively; the synapses specified in these rules are budded from the neuron labelled with $c_{(1,t)}$. The latter pair of rules generates two neurons with labels $c_{(2,f,f)}$ and $c_{(2,t,f)}$, respectively; the synapses mentioned in these rules go from the neuron labelled with $c_{(1,f)}$ to the newly created neurons. In the meanwhile the truth assignments (f,t), (t,t), (f,f), (t,f), for the Boolean variables x_1 and x_2, are generated at each leaf node, as illustrated in Figure 5. Since the interaction environment of neurons $\sigma_{c_{(1,t)}}$ and $\sigma_{c_{(1,f)}}$ has changed, the rules applied in this step cannot be applied anymore to these neurons.
By continuing in this way, by applying the budding rules of type a_1) in a maximally parallel way for n computation steps, a complete binary tree of depth n having 2^n leaves (hence an exponentially large workspace) is built. The label of each leaf node encodes a truth assignment of length n, hence all possible truth assignments for the Boolean variables $x_1, x_2, \ldots, x_n$ are generated.

Phase B. The pre-computation to construct the SN P system structure continues until it converges to the output neuron in a further few steps. The main goal of this part of the construction algorithm is to design the substructure which is devoted to test the satisfiability of the clauses of the instance γ_n of SAT(n, m) given as input against all possible truth assignments, and to determine whether there exist some assignments that satisfy all the clauses of γ_n.

The substructure is composed of 2^n subsystems, each being a sequence of n neurons $\sigma_{c_{(j,x_n)}}, 1 \leq j \leq n$, including the leaf nodes of the dendritic tree. A subsequence $x_n = (x_{n1}, x_{n2}, \ldots, x_{nn}) \in \{t, f\}^n$ in a neuron label $c_{(j,x_n)}$ represents a truth assignment, and we can abstractly assign a pair $(j, x_n(j))$ to a neuron $\sigma_{c_{(j,x_n)}}$ as its identity. Thus each subsystem represents a truth assignment formed by its neurons' identities. As stated above, a neuron with identification $(j_1, x(j_1) = t)$ has 3 synapses coming from the generator module, whereas a neuron with identity $(j_2, x(j_2) = f)$ is connected with the generator by means of 4 synapses. As we will see, these connections are used to perform assignments to the Boolean variables $x_1, x_2, \ldots, x_n$ that compose γ_n, and to check which assignments satisfy the clause of γ_n currently under consideration.

For instance, the case in which $n = 2$ is described in Figure 5, where $2^2 = 4$ different truth assignments of length 2 have been generated for the two Boolean variables x_1 and x_2. The first subsystem is composed of two neurons having labels $c_{(2,f,t)}$ and $c_{(1,f,t)}$, respectively. The former is associated with the Boolean value *false*, as $x_2 = (f, t)$ and $x_2(2) = f$, while the latter is associated with *true*, as $x_2(1) = t$; altogether they form the truth assignment (f, t). The other subsystems are similar, and are associated with the truth assignments (t, t), (f, f) and (t, f). One can see that the four truth assignments are well distinguished from each other by the layer structure of the four subsystems.

To build the substructure of n layers mentioned above, from now on two rules of types $\mathbf{a_2}$) and $\mathbf{a_3}$) are applied simultaneously to a same neuron for $n - 1$ steps. The first rule creates a new neuron with an associated synapse, while the second rule creates 3 or 4 synapses to the generator block. The same process occurs during the n-th step, by means of the rules of types $\mathbf{a_3}$) and $\mathbf{a_4}$); note that in this step the rules of type $\mathbf{a_2}$) cannot be applied anymore.

$\mathbf{a_2}$) $(c_{(n+1-j,x_n)}, c_{(n-j,x_n)})[\ ]_{c_{(n-j,x_n)}} \rightarrow (c_{(n-j,x_n)}, c_{(n-1-j,x_n)})[\ ]_{c_{(n-1-j,x_n)}}$,
$p \in \{t, f\}$, $0 \leq j \leq n-1$, $1 \leq k \leq n$,
$c_{(k,0,x_k^{(p)})} = c_{(k-1,n,x_{k-1})}$, $x_k^{(p)} = (p, x_{k-1}) \in \{t, f\}^k$.

$\mathbf{a_3}$) $(c_{(n+1-j,x_n)}, c_{(n-j,x_n)})[\ ]_{c_{(n-j,x_n(j+1)=p)}} \rightarrow (c_{(n-j,x_n(j+1)=p)}, e_i)[\ ]_{e_i}$,
$0 \leq j \leq n$, $p \in \{t, f\}$ and $s \leq i \leq 3$, where $s = 1$ if $p = t$, and $s = 0$ if $p = f$,
$c_{(n,0,x_n)} = c_{(n-1,n,x_n)}$.
We are now in the $(n + 1)$-th step of the computation. When $j = 0$, both rules of types $\mathbf{a_2}$) and $\mathbf{a_3}$) are applicable to each neuron $\sigma_{c_{(n,x_n)}}$ of layer n. The former rules generate neurons $\sigma_{c_{(n-1,x_n)}}$ with a synapse $(c_{(n,x_n)}, c_{(n-1,x_n)})$. The latter type of rules creates three synapses to all neurons of type $\sigma_{c_{(n,x_n(1)=t)}}$ coming from the neurons $\sigma_{c_{e_i}}$, $1 \leq i \leq 3$, and four synapses to the neurons

$\sigma_{c_{(n,x_n(1)=f)}}$ coming from the four neurons $\sigma_{c_{e_i}}$, $0 \leq i \leq 3$, of the generator block. The neuron budding rules of type $\mathbf{a_2}$) and the synapse creation rules of type $\mathbf{a_3}$) are applied simultaneously to the same neurons (leaf nodes) in layer n in the following $n-1$ steps, since their interaction environments coincide. The effect of the application of these rules is the production of neurons having connections with the generator block.

So doing, 2^n subsystems, each one composed of a sequence of n neurons, are generated starting from layer n. In each subsystem, every neuron corresponding to the Boolean value *true* ($x_n(j) = t$) is connected with the generator block by means of three synapses, while the neurons that correspond to the Boolean value *false* ($x_n(j) = f$) are connected with the generator block by four synapses.

From the $(2n+1)$-th step of the computation on, no interaction environment of any neuron in the system allow to activate the rules of type $\mathbf{a_2}$). Hence these rules cannot be applied, but the computation continues with the next types of rules.

$\mathbf{a_4}$) $(c_{(2,x_n)}, c_{(1,x_n)})[\ ]_{c_{(1,x_n)}} \rightarrow (c_{(1,x_n)}, c_1)[\ ]_{c_1}$.

The rules of type $\mathbf{a_4}$) can be applied in parallel to the leaf nodes (neurons) of layer n; they produce the neurons σ_{c_1} forming the $(n+1)$-th layer and, meanwhile, the rules of type $\mathbf{a_3}$) create synapses from these neurons to the generator block.

$\mathbf{a_5}$) $(c_{(1,x_n)}, c_1)[\ ]_{c_1} \rightarrow (c_1, c_2)[\ ]_{c_2}$,

$\mathbf{a_6}$) $(c_{(1,x_n)}, c_1)[\ ]_{c_1} \rightarrow (b_i, c_1)[\ ]_{b_i}$, $1 \leq i \leq 3$.

While the rules of type $\mathbf{a_5}$) are applied to the neurons σ_{c_1} and bud neurons σ_{c_2}, the rules of type $\mathbf{a_6}$) are also applied and create three synapses coming from the neurons σ_{b_i}, $1 \leq i \leq 3$, to each neuron σ_{c_1}. In this way, layer $n+2$ is formed.

$\mathbf{a_7}$) $(c_1, c_2)[\ ]_{c_2} \rightarrow (c_2, c_3)[\ ]_{c_3}$,

$\mathbf{a_8}$) $(c_1, c_2)[\ ]_{c_2} \rightarrow (c, c_2)[\ ]_{c}$.

The rules of types $\mathbf{a_7}$) and $\mathbf{a_8}$) apply simultaneously to every neuron σ_2 having a synapse (c_1, c_2). As a result, a new neuron σ_{c_3} is budded with a connection (c, c_2) coming from neuron σ_c. All the neurons σ_{c_2} in the same layer are subjected to the same effect, since the rules are applied in the maximally parallel way.

$\mathbf{a_9}$) $(c_2, c_3)[\ ]_{c_3} \rightarrow (c_3, out)[\ ]_{out}$.

The pre-computation of the SN P system structure is completed by forming the connections from the neurons σ_{c_3} to the output neuron σ_{out}, by means of the rules of type $\mathbf{a_9}$). These rule are applied in the maximally parallel way to all the neurons in layer $n+3$.

Summarizing, phases A and B build an empty (that is, containing no spikes) structure of an SN P system, that can be used to solve all the instances of SAT(n, m) in a linear (with respect to n) number of computation steps. The size of the structure is exponential with respect to n.

Solving SAT *(Phase C).*

Given an instance γ_n of SAT(n, m), we first encode it as a sequence of spike variables, as explained in Section 3, equation (1). Then, the computation of the system may start. The sequence of spikes encoding γ_n is introduced in the system, using neuron σ_{soma}. Let us see at what spiking rules are used to compute the solution, with a brief description for each.

c$_1$) $[a \to a]_{c_{(i,x_i)}}$; $[a^2 \to a^2]_{c_{(i,x_i)}}$
$0 \le i \le n,\ x_i \in \{t, f\}^i,\ c_{(0,x_0)} = soma,$

c$_2$) $[a \to a; n-1]_s$.
We insert 0, 1 or 2 spikes into the system by rule **c$_1$**) using the input neuron σ_{soma}, according to the value of the spike variable α_{ij} we are considering in the representation of γ_n. In the meanwhile we insert a single spike a into neuron σ_s, to fire once the rule **c$_2$**), thus activating the generator block.

Each spike, encoding a spike variable inserted into the input neuron, is duplicated and transmitted to the next layer of neurons. This duplication is performed n times, until 2^n replicated copies of the spike are placed in the leaf nodes (in layer n) of the dendritic tree.

c$_3$) $[a \to a]_{e_i}; 0 \le i \le 3,$
$[a \to a; n-1]_+$; $[a \to a; n-1]_-$.
These are the spiking rules of the generator block. Each n steps, the generator provides 3 and 4 spikes, respectively, to the neurons of layer n associated with the truth values t and f. This is made in order to test the satisfiability of a clause which has propagated through the layers of the dendritic tree, by checking it against all possible truth assignments to the variables $x_1, x_2, \ldots, x_n$.

In another n steps, the 2^n copies of the clause of γ_n take place in the corresponding subsystems located in layers from $n+1$ to $2n$, where the satisfiability of the clause against all possible truth assignments is tested. For this purpose, the spike-truth values a^4 and a^3 are assigned from the generator to the spike-variables of the clause, according to the truth assignments represented by the neurons that compose the subsystems. In fact, recall that in each subsystem every neuron corresponding to the Boolean value *true* $(x_n(j) = t)$ is connected with the generator block by means of three synapses, while the neurons that correspond to the Boolean value *false* $(x_n(j) = f)$ are connected with the generator by means of four synapses. The satisfiability is then checked by means of the rules of types **c$_4$**) and **c$_5$**) residing in the neurons.

c$_4$) $[a \to a]_{t_t}$; $[a^3 \to \lambda]_{t_t}$; $[a^2 \to a^2]_{t_1}$;
$[a^4 \to a]_{t_t}$; $[a^5 \to \lambda]_{t_t}$; $[a^2 \to a]_{t_0}$;
$t_t = c_{(j,x_n(j)=t)}, 1 \le j \le n,$
$t_1 = c_{(j,x_n(j)=t)}, 2 \le j \le n,$
$t_0 = c_{(1,x_n(n)=t)}, x_n \in \{t, f\}^n.$

These are the spiking rules that reside in the neurons of layer n, associated with the Boolean value *true* (in Figure 5 $n = 2$, $\sigma_{c_{(2,t,f)}}$ stands for $false$ while $\sigma_{c_{(1,t,f)}}$ stands for *true*). The rules $a^2 \to a^2$, $a^2 \to a$, and $a \to a$ are used to transmit the spike variables a, a^2 along the subsystems. Once a clause C_i is ready to be tested for satisfiability, each neuron associated with *true* contains either one spike (a), two spikes (a^2) or is empty (λ). As a spike variable a represents the occurrence of a Boolean variable x_j in C_i, to which a *true* value (a^3) sent by the generator is assigned, resulting in a *yes* answer (a^4), then it passes to the neuron σ_{c_1} along the subsystem as an indication that C_i is satisfied by a truth assignment in which the Boolean variable x_j is true. On the other hand, if the Boolean value *true* (a^3) is assigned to a spike variable that represents the occurrence of $\neg x_j$ in C_i (a^2) or the fact that x_j does not occur in C_i (λ), then in these cases the answer is *no*, which is computed by the rules $a^3 \to \lambda$ and $a^5 \to \lambda$.

c₅) $[a \to a]_{f_f}$; $[a^4 \to \lambda]_{f_f}$; $[a^2 \to a^2]_{f_1}$;
$[a^5 \to \lambda]_{f_f}$; $[a^6 \to a]_{f_f}$; $[a^2 \to a]_{f_0}$;
$f_f = c_{(j,x_n(j)=f)}, 1 \leq j \leq n$,
$f_1 = c_{(j,x_n(j)=f)}, 2 \leq j \leq n$,
$f_0 = c_{(1,x_n(n)=f)}, x_n \in \{t, f\}^n$.
These are the spiking rules that reside in the neurons of layer n, associated with the Boolean value *false*. The functioning of these rules is similar to that of rules **c₄**).

c₆) $[a \to a; n-1]_{b_i}; 1 \leq i \leq 3$,
$[a^2/a \to a]_{c_1}$; $[a^3 \to \lambda]_{c_1}$;
$[a^4 \to a]_{c_1}$; $[a^5 \to a]_{c_1}$.
Whether an assignment satisfies or not the clause under consideration, is checked by a combined functioning of the neurons with label 1 in layer $n+1$ and the neurons with label $b_i, 1 \leq i \leq 3$, in the generator.

c₇) $[a \to \lambda]_{c_2}$; $[a^2 \to a]_{c_2}$;
$[a \to a]_c$.
With a combined action of neuron σ_c, neuron σ_{c_2} sends a spike to neuron σ_{c_3} if and only if the corresponding assignment satisfies the clause under consideration.

c₈) $[a^m \to a]_{c_3}$;
$[a^+/a \to a]_{out}$.
Neurons with label c_3 count how many clauses of the instance γ_n are satisfied by the corresponding truth assignments. If one of these neurons get m spikes, then it fires. Hence the number of spikes that reach neuron *out* is the number of assignments that satisfy all the clauses of γ_n. The output neuron fires if it contains at least one spike, thus signalling that the problem has a positive solution; otherwise, there is no assignment that satisfies the instance γ_n.

This stage of computation ends at the $(nm+n+4)$-th step. The entire computation of the system thus halts in at most $nm + n + 5$ computation steps, hence in a polynomial time with respect to n and m.

In conclusion, we obtained a deterministic, polynomial time and uniform solution to SAT(n, m) in the framework of SN P systems.

5 Conclusions and directions for future research

In the present paper we proposed a way to solve the **NP**-complete decision problem SAT in a polynomial time with respect to the number n of Boolean variables and the number m of clauses that compose the instances of SAT being solved. Specifically, we introduced SN P systems with *neuron budding rules*, a new feature that enhances the efficiency of SN P systems by allowing them to generate an exponential size synapse graph (regarded as the workspace of the system) in a polynomial time with respect to n.

Neuron budding rules drive the mechanism of neuron production and synapse creation, according to the interaction of neurons with their neighbourhoods (described by the synapses that connect them to other neurons). We have shown that a very restricted type of neuron budding rules, involving one or two synapses (actually, when two synapses are involved, they appear one in each side of the rule) is sufficient to solve the SAT problem. The solution is computed in two stages: the first phase builds an exponential size SN P system that contains no spikes; then, this SN P system is fed with the instance of SAT to be solved (encoded in an appropriate way) and the answer is computed. The system works in the deterministic and maximally parallel manner.

The idea of producing new neurons in SN P systems is not new: already in [15] neurons are generated by *division*. However, both biological motivation and mathematical formal definition are different: neuron budding in this paper depends on the connections (structure) with other neurons, while neuron division depends on the number of spikes occurring inside the neurons (that is, the contents); hence they are two different ways to increase the workspace of SN P systems.

An open question is whether SN P systems with budding rules can be used to efficiently solve other computationally difficult problems, such as *numerical* **NP**-complete problems and **PSPACE**-complete problems.

SN P systems with neuron budding rules can be extended by introducing more general rules, which in some sense capture the dynamic interaction of neurons with their neighbourhood. One possible form of such general rules is as follows: $A_i[\,]_i B_i \to C_j[\,]_j D_j$, where A_i, B_i and C_j, D_j are the sets of synapses coming to and going out from, respectively, the specified neurons σ_i and σ_j. Clearly, in such general rules, more than one synapses can be involved in the neighbourhood of the considered neuron.

Acknowledgments

The work of Tseren-Onolt Ishdorj was supported by BIOTARGET, a joint project between the University of Turku and Åbo Akademi University, funded by the Academy of Finland. The work of L. Pan was supported by the National Natural Science Foundation of China (Grant Nos. 60674106, 30870826, 60703047, and 60533010), Program for New Century Excellent Talents in University (NCET-05-0612), Ph.D. Programs Foundation of Ministry of Education of China (20060487014), Chenguang Program of Wuhan (200750731262), HUST-SRF (2007Z015A), and by the Natural Science Foundation of Hubei Province (2008CDB113 and 2008CDB180). The work of Alberto Leporati was partially supported by MIUR project "Mathematical aspects and emerging applications of automata and formal languages" (2007).

References

1. H. Chen, R. Freund, M. Ionescu, Gh. Păun, M.J. Pérez-Jiménez: On string languages generated by spiking neural P systems. *Fundamenta Informaticae* **75**:141–162, 2007.
2. H. Chen, M. Ionescu, T.-O. Ishdorj: On the efficiency of spiking neural P systems. *Proceedings of the* 8th *International Conference on Electronics, Information, and Communication*, Ulanbator, Mongolia, June 2006, pp. 49–52.
3. H. Chen, M. Ionescu, T.-O. Ishdorj, A. Păun, Gh. Păun, M.J. Pérez-Jiménez: Spiking neural P systems with extended rules. *Fourth Brainstorming Week on Membrane Computing* (M.A. Gutiérrez-Naranjo, Gh. Păun, A. Riscos-Núñez, F.J. Romero-Campero, eds.), vol. I, RGNC Report 02/2006, Research Group on Natural Computing, Sevilla University, Fénix Editora, 2006, pp. 241–266.
4. W. Gerstner, W. Kistler: *Spiking neuron models. Single neurons, populations, plasticity*. Cambridge University Press, 2002.
5. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae* **71**(2–3):279–308, 2006.
6. T.-O. Ishdorj, A. Leporati: Uniform solutions to SAT and 3-SAT by spiking neural P systems with pre-computed resources. *Natural Computing* **7**(4):519–534, 2008.
7. T.-O. Ishdorj, A. Leporati, L. Pan, X. Zeng, X. Zhang: Deterministic solutions to QSAT and Q3SAT by spiking neural P systems with pre-computed resources. Submitted for publication.
8. M.R. Garey, D.S. Johnson: *Computers and intractability. A guide to the theory on* **NP***–completeness*. W.H. Freeman and Company, 1979.
9. A. Leporati, M.A. Gutiérrez-Naranjo: Solving Subset Sum by spiking neural P systems with pre-computed resources. *Fundamenta Informaticae* **87**(1):61–77, 2008.
10. A. Leporati, G. Mauri, C. Zandron, Gh. Păun, M. J. Pérez-Jiménez: Uniform solutions to SAT and Subset Sum by spiking neural P systems. *Natural Computing* (Online version), DOI: 10.1007/s11047-008-9091-y.
11. A. Leporati, C. Zandron, C. Ferretti, G. Mauri: Solving numerical NP-complete problem with spiking neural P systems. In: G. Elefterakis et al. (Eds.), *Membrane Computing,* 8th *International Workshop (WMC 8)*, Revised Selected and Invited Papers. LNCS 4860, Springer, 2007, pp. 336–352.

12. A. Leporati, C. Zandron, C. Ferretti, G. Mauri: On the computational power of spiking neural P systems. *International Journal of Unconventional Computing* **5**(5):459–473, 2009.
13. W. Maass: Computing with spikes. *Special Issue on Foundations of Information Processing of TELEMATIK* **8**(1):32–36, 2002.
14. W. Maass, C. Bishop (eds.): *Pulsed neural networks.* MIT Press, 1999.
15. L. Pan, Gh. Păun, M. J. Pérez-Jiménez: Spiking neural P systems with neuron division and budding. *Seventh Brainstormin Week on Membrane Computing*, (R. Gutiérrez-Escudero, M.A. Gutiérrez-Naranjo, Gh. Păun, I. Pérez-Hurtado, A. Riscos-Núñez, eds.), vol. II, RGNC Report 01/2009, Research Group on Natural Computing, Sevilla University, Fénix Editora, 2009, pp. 151–167.
16. Gh. Păun: *Membrane computing – An introduction.* Springer–Verlag, Berlin, 2002.
17. Gh. Păun: Twenty six research topics about spiking neural P systems. *Fifth Brainstorming Week on Membrane Computing* (M.A. Gutiérrez-Naranjo, Gh. Păun, A. Romero-Jiménez, A. Riscos-Núñez, eds.), RGNC Report 01/2007, Research Group on Natural Computing, Sevilla University, Fénix Editora, 2007, pp. 263–280.
18. M. Sipser: *Introduction to the theory of computation.* PWS Publishing Company, Boston, 1997.
19. The P systems Web page: `http://ppage.psystems.eu/`
20. Think and Grow Toys: `http://www.tagtoys.com/dendrites.php`