About the Efficiency of Spiking Neural P Systems

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

[1] 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
[2] Computational Biomodelling Laboratory
Åbo Akademi University
Department of Information Technologies
20520 Turku, Finland
tishdorj@abo.fi
[3] Research Group on Natural Computing
Department of CS and AI, University of Sevilla
Avda Reina Mercedes s/n, 41012 Sevilla, Spain
Institute of Mathematics of the Romanian Academy

Summary. Spiking neural P systems were proved to be Turing complete as function computing or number generating devices. Moreover, it has been considered in several papers that spiking neural P systems are also computationally efficient devices working in a non-deterministic way or with exponential pre-computed resources. In this paper, neuron budding rules are introduced in the framework of spiking neural P systems, which is biologically inspired by the growth of dendritic tree of neuron. Using neuron budding rules in SN P systems is a way to trade space for time to solve computational intractable problems. The approach is examined here with a deterministic and polynomial time solution to SAT problem without using exponential pre-computed resources.

1 Introduction

Computational efficiency of spiking neural P systems (in short, SN P systems) has been investigated in a series of works [1, 6, 8, 9, 10], recently. In the framework of SN P systems, most of the solutions to computationally hard problems are based on non-determinism [9, 10, 11] or exponential pre-computed resources [1, 6, 8, 7]. The present paper proposes a rather different way to address this issue in a sense that no pre-computed resource is used but it is computed by a SN P system.

It has been claimed in [11] that an SN P system of polynomial size cannot solve in a deterministic way in a polynomial time an **NP**-complete problem (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 (pre-computed resources, ways to exponentially grow the workspace during the computation, non-determinism, and so on).

A possibility of using spiking neural P systems for solving computationally hard problems, under the assumption that some (possibly exponentially large) pre-computed resources are given in advance has been presented in [6]. Specially, in [6], a uniform family of spiking neural P systems was proposed which can be used to address the **NP**-complete problems, in particular, to solve all the instances of SAT which can be built using n Boolean variables and m clauses, in a time which is quadratic in n and linear in m.

In the present paper, we continue the study considered in [6] and particularly focus on a possible way to construct an SN P system such that the system can compute the necessary resources (exponentially large work space) to be used in advance by itself. For this purpose, we extend the SN P systems [6] by introducing neuron budding rules. We show that the SN P systems with budding rules can (pre-)compute the exponential work space in polynomial time with respect to the size of the instances of the problem we want to solve, however, the problem is solved too by the same system. All the systems we will propose work in a *deterministic* way.

The biological motivation of the mechanism for expanding the work space (net structure) of SN P systems by introducing neuron budding comes from the growth of dendritic tree of neural cells [15]. The brain is made up of about 100 billion cells. Almost all brain cells are formed before birth. Dendrites (from Greek, tree) are the branched projections of a neuron. The point at which the dendrites from one cell contact the dendrites from another cell is where the miracle of information transfer (communication) occurs. *Brain cells can grow as many as 1 million billion dendrite* connections – a universe of touch points. The greater the number of dendrites, the more information that can be processed. Dendrites grow as a result of stimulation from and interaction with the environment. With limited stimulation there is limited growth. With no stimulation, dendrites actually retreat and disappear. These microscope photographs illustrated in Figure 1 show actual dendrite development. Dendrites begin to emerge from a single neuron (brain cell) developing 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 considered as abstract neurons and the branches of dendrite tree are consider as abstract synapses. The new connection between dendrites from two different neuron cells is understood as new created synapses. In this way, new neurons and synapses can be produced during the growth of dendrite tree.

The formal definition of neuron budding rule and its semantics will be given in Section 2.

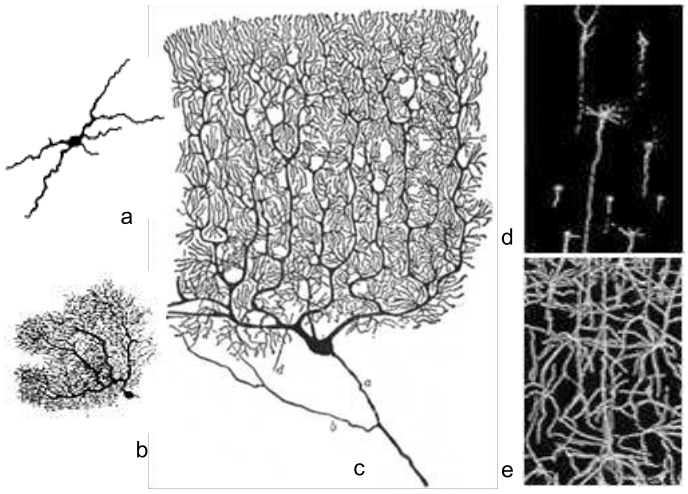

Fig. 1. Growing neuron: a. dendrites begin to emerge from a single neuron, b. developed into a cluster of touch points; c. Ramon y Cajal, Santiago. Classical drawing: Purkinje cell; d. newborn neuron dendrites, e. 3 months later. Photos from Tag Toys [15]

2 SN P systems with neuron budding rules

A *spiking neural P system with neuron budding* 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) *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 $c \geq 1$, $p \geq 0$, $d \geq 0$, with the restriction $c \geq p$;
   (2) *neuron budding rule* $x[\ ]_i \rightarrow y[\ ]_j$, where $x \in \{(k, i), (i, k), \lambda\}$, $y \in \{(i, j), (j, i)\}$, $i, j, k \in H$, $i \neq k$, $i \neq j$.
6. $in, out \in H$ indicate the *input* and the *output* neurons of Π.

The way of presentation of SN P system is here slightly different from the usual definition present in the literature, where the neurons presented initially in the system are explicitly listed as $\sigma_i = (n_i, R_i)$, $1 \leq i \leq m$ and R_i are the rules

associated with neuron with label i. In what follows we will refer to neuron with label $i \in H$ also denoting it with σ_i.

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 *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*.

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 it 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.

If a neuron σ_i has only synapse x where $x \in \{(i, k), (k, i), \lambda\}$, $i \neq k$, then rule $x[\,]_i \to y[\,]_j$ is enabled and can be applied, where $y \in \{(i, j), (j, i)\}$. The synapse x describes the interaction environment of neuron σ_i with another neuron. As a result of the rule application, a new neuron σ_j and a synapse y are established provided that they do not exist already; if a neuron with label j does already exist 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 depends on the environment of the associated neuron, instead of the spikes contained in the associated neuron; a budding rule can be applied only if the associated neuron has the environment exactly as the rule described; in other words, even if the environment has a proper sub-environment that enables a budding rule, but the whole environment does not enables the budding rule, then the rule cannot be applied. The rules of such type are applied in a maximal parallel way: if the environment of neuron σ_i enables several budding rules, then all these rules are applied; 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 the usual way in P systems with cell division or cell creation, where at most one rule division rule or creation rules can be applied to one membrane or one object, respectively.

In each time unit, if a neuron σ_i can use one of its rules, then a rule from R *must* be used. If several spiking rules are enabled in neuron σ_i, then only one of them is chosen non-deterministically. If the environment of neuron σ_i enables several budding rules, then all these rules are applied. If both spiking rules and budding rules are enabled in the same step, then one type of rules is chosen non-deterministically. When a spiking rule is used, the state of neuron σ_i (open or closed) depends on the delay d. When a neuron budding rule is applied, at this step the associated neuron is closed, it cannot receive spikes. In the next step, the neurons obtained by budding will be open and can receive spikes.

The *configuration* of the system is described by the topology structure of the system, the number of spikes associated with each neuron, and the *state* of each neuron (open or closed). 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 neurons are open and no rule can be used.

In the following, we give an example to make the usage of budding rules transparent, where neither spike nor spiking rule is of interest.

An example. The system Π_1 has initial topological structure shown in Figure 2(a), and the budding rules $(1,3)[\,]_3 \to (3,4)[\,]_4$, $(1,3)[\,]_3 \to (3,5)[\,]_5$, $(2,1)[\,]_2 \to (6,2)[\,]_6$, $(3,4)[\,]_4 \to (4,7)[\,]_7$ and $(6,2)[\,]_6 \to (6,5)[\,]_5$.

In the initial topological structure, neuron σ_3 has two synapses $(1,3)$ and $(2,1)$, and no other synapses are associated with it; as the environment of neuron σ_3 enables both rules $(1,3)[\,]_3 \to (3,4)[\,]_4$ and $(1,3)[\,]_3 \to (3,5)[\,]_5$, the rules are applied in the maximal parallel application manner. As a result, two new neurons σ_4 and σ_5, and two synapses $(3,4)$ and $(3,5)$ are produced. At the same time, the rule $(2,1)[\,]_2 \to (6,2)[\,]_6$ is applied to neuron σ_2 with a synapse $(2,1)$, thus, neuron σ_6 and synapse $(6,2)$ are produced. The structure is shown in 2(b) after step 1.

At the second step, the rules $(1,3)[\,]_3 \to (3,4)[\,]_4$ and $(1,3)[\,]_3 \to (3,5)[\,]_5$ cannot apply again as the two newly created synapses $(3,4)$ and $(3,5)$ going out from neuron σ_3 have changed the environment of it. Similarly, the rule $(2,1)[\,]_2 \to (6,2)[\,]_6$ cannot be used again. As neuron σ_4 has only synapse $(3,4)$, its environment enables the rule $(3,4)[\,]_4 \to (4,7)[\,]_7$ to be applied to it, then a new neuron σ_7 and a synapse $(4,7)$ are produced. Neuron σ_6 has only synapse $(6,2)$, then rule $(6,2)[\,]_6 \to (6,5)[\,]_5$ is enabled and applied. Since a neuron with label 5 already exist, no new neuron with label 5 is budded instead, a synapse $(6,5)$ to neuron σ_5 from neuron σ_6 is established, this is the principle of neuron budding rules. The corresponding structure is shown in Figure 2(c). Now no rule is enabled by any neuron interaction environment, thus the system halts.

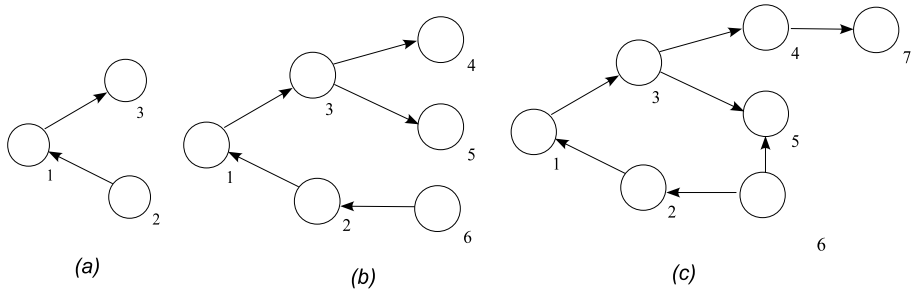

Fig. 2. Structure transition of SN P system Π_1

3 Brief of pre-computed SN P systems solving SAT

As we mentioned in Section 1, a way to solve **NP** hard problems by SN P systems is to assume an exponential working space has been pre-computed in advance, based on that given work space a family of SN P systems solves all the possible instances of the problem in polynomial time, [6].

Let us recall here the basic description of SAT (satisfiability) problem, a well know **NP**-complete problem. 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 $x_1, x_2, \ldots, x_n$. 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 gives 1 (*true*) as a result.

Let us denote by $\text{SAT}(n, m)$ 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 $\text{SAT}(n, m)$ in a number of steps which is quadratic in n and linear in m.

Let us first briefly summarize here the overview of the considered system $\Pi_{SAT}(\langle n, m\rangle)$ from [6], and its structure and functioning that solves all the possible instances of $\text{SAT}(n, m)$.

The system structure is composed by $n + 5$ layers, see Figure 3. The first layer (numbered by 0) is composed by a single input neuron, that is used to insert the representation of the instance $\gamma_n \in \text{SAT}(n, m)$ to be solved. Note that layer 1, as well as the subsequent $n - 1$ layers, is composed by a sequence of n neurons, so that the layer contains the representation of one clause of the instance. In layer n, we have got 2^n copies of the subsystem; each subsystem contained in this layer is bijectively associated to one possible assignment to variables $x_1, x_2, \ldots, x_n$. Simply say, the neurons in a subsystem are two types: f and t; the types indicate that the corresponding Boolean variable is assigned with the Boolean values $t(rue)$ or $f(alse)$, respectively. However, the all subsystems of layer n are injectively distin-

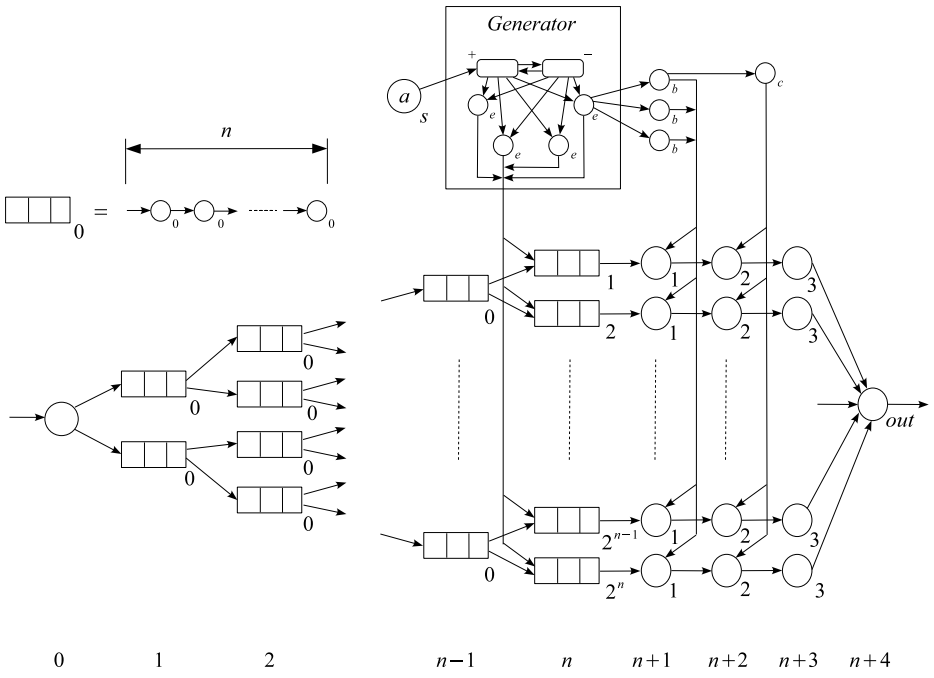

Fig. 3. SN P system structure which solves SAT(m,n). From [6].

guished from each other with respect to the all possible different truth assignments for variables $x_1, x_2, \ldots, x_n$ represented by each subsystem. The subsystems that occur in layer n together with the so called *generator* have a very specific functions such that all possible assignments are tested in here in parallel against the 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 from the generator. This means that the generator have three synapses going to neurons t and four synapses towards neurons f.

Those assignments that satisfy the clause produce a (single) spike in the corresponding neuron 2 (that occurs in the same row, in layer $n+2$), which is accumulated in the associated neuron 3, that operates like a counter. When the first clause of γ_n has been processed, the second takes place in the subsystems in layer n in n steps, and all possible assignments are tested, etc. When all the m clauses of γ_n have been processed, neurons 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 *out*, thus signalling that their corresponding assignment satisfies all the clauses of the instance. Neuron *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.

In the next section, in particular, we aim to show the fact that the *assumed* pre-computed work space used in [6] to solve SAT *can be* pre-computed practically in advance in polynomial time by SN P systems with budding rules. Then, a solution to SAT problem is given by the systems with already pre-computed work space.

4 Uniform solution to SAT by (dendritic) SN P systems

Our SN P system with budding rules is composed of two subsequent subsystems: construction of a SN P system structure which meant to solve SAT problem uniformly and the SN P systems family, [6], which solves the SAT problem efficiently – for the sake of simplicity, we avoid the neuron budding and the spike firing rules are used at the same time in each subsystem.

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

where:

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

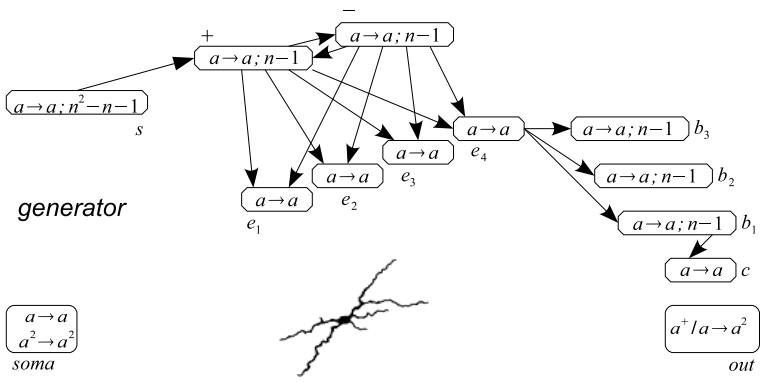

Fig. 4. The initial topological structure (new born dendrite) of the system Π: *soma* and *out* neurons, *generator*.

Constructing the system structure

The system initially contains an input neuron σ_{soma}, an output neuron σ_{out}, and a sub-structure so-called generator block G composed of the set of neurons Σ and the set of synapses syn_0, $|\Sigma| = |syn_0| = 13$, the corresponding topological structure is illustrated in Figure 4.

The generation mechanism is governed by only neuron budding rules and controlled by the labels of budding neurons and the created synapses. The labels of each neuron in a subsystem in layer n encodes an associated truth assignment.

The system construction algorithm consists of two main parts:

A. To generate the *dendritic-tree* sub-structure (the layers 0 – n in Figure 3, exponentially large in n) and the truth assignments for n Boolean variables. The process starts from the initial neuron σ_{soma} (the root node).

B. To complete the network structure. The subsystems in nth layer of the system establish connections to the *generator block* according to the truth assignments represented in those subsystems, and they are expanded by further three layers, finally converged to the output neuron σ_{out}.

A. The dendritic-tree generation process, controlled by the labels of neurons as well as the synapses, starts from the initial neuron σ_{soma} (cell body). It is noteworthy that since the truth assignments associated with the subsystems in nth layer are encoded in the labels of those neurons compose each subsystem, the truth assignments are being generated while the dendritic-tree has been constructed.

The label of a neuron σ_c is a sequence of the form

$$c = (k, j, x_k^{(p)}) = (k, j, x_k(1) = p) = (k, j, p, x_{k2}, \ldots, x_{kk}),$$

$p \in \{t, f\}$, where the first pair (k, j) indicates the location of the neuron on the dendritic-tree: k is the layer number, j is the place where the neuron is in its subsystem, the subsequence $x_k^{(p)}$ represents a string of length k formed by Boolean values t and f being generated. Whereas p in $x_k^{(p)}$ indicates that the first entry of the subsequence is exactly p – which is later importantly used in the budding rules to distinguish the being generated truth assignments from a same neuron. Moreover, we stress again that the labels of neurons of a chain of length k in a layer k represents a truth-assignment v of length k, precisely, v is a sequence formed by $x_k(j)$, $1 \leq j \leq k$, of the neuron labels $c = (k, j, x_k)$ of a chain. However, hence each chain or subsystem of a layer structure is a separate unit and associates with a truth assignment, all the truth assignments represented in a layer are distinguishable from each other. In other words, the truth assignments are encoded in both the labels of neurons of the chains and its layer structure of composing units too. We do not care which assignment is associated with which subsystem of the layer.

In this phase of computation three types of budding rules are performed for the role: budding rules of type $\mathbf{a_0}$) applied to the neuron σ_{soma} which initiates the generation of the structure; the *dendritic-tree* structure is constructed from the layer 0 towards the layer n, for each layer two types of rules such as $\mathbf{a_1}$) $n - 1$ times and $\mathbf{a_2}$) once, are alternated, total $n \times n$ steps needed to complete.

a₀) $[\ ]_{c_{soma}} \to (c_{soma}, c_{(1,1,t)})[\ ]_{c_{(1,1,t)}}$,
$[\ ]_{c_{soma}} \to (c_{soma}, c_{(1,1,f)})[\ ]_{c_{(1,1,f)}}$,
where $(c_{soma}, c_{(1,1,t)}), (c_{soma}, c_{(1,1,f)}) \in syn$.
The initial neuron $\sigma_{c_{soma}}$ buds two new neurons as the rules a_0) apply to it simultaneously. The newly produced neurons are: $\sigma_{c_{(1,1,t)}}$ with a synapse $(c_{soma}, c_{(1,1,t)})$ coming from the father neuron and $\sigma_{c_{(1,1,f)}}$ connected with the father neuron by a synapse $(c_{soma}, c_{(1,1,f)})$, respectively. Where the symbols t and f in the neuron labels indicate truth values $t(rue)$ and $f(alse)$, respectively, hence the two truth assignments (t) and (f) of length 1 for a single Boolean variable y_1 are formed. Note that the left hand side of each rule a_0) (where $\lambda \in syn_0$ is omitted) requires its interaction environment is empty i.e no synapse exists connected to the neuron $\sigma_{c_{soma}}$. Once the rules have applied, the interaction environment of the neuron $\sigma_{c_{soma}}$ has been evolved having two new synapses going out are created, which makes those rules are not applicable to this neuron anymore. Thus, the base of the first layer of the dendritic-tree has been established, at the first step of the computation.
An almost complete system structure for SAT$(2, m)$ is depicted in Figure 5, which is worth to follow during the construction.
To complete the established layer 1 (in general, i, $1 \le i \le n$), the rules of type a_1) generate the 2 (in general 2^i number of) subsystems or the chains of n neurons.

a₁) $(c_{(k,j-1,x_k^{(p)})}, c_{(k,j,x_k^{(p)})})[\ ]_{c_{(k,j,x_k^{(p)})}} \to (c_{(k,j,x_k^{(p)})}, c_{(k,j+1,x_k^{(p)})})[\ ]_{c_{(k,j+1,x_k^{(p)})}}$,
$p \in \{t, f\}$, $1 \le j \le n-1$, $1 \le k \le n$, $c_{(k,0,x_k^{(p)})} = c_{(k-1,n,x_{k-1})}$, $x_k^{(p)} = (p, x_{k-1}) \in \{t, f\}^k$.
The chains composed of n neurons in a layer k are generated by iterative applications of the rules of type a_1) in $n-1$ steps. This rule can be applied in a neuron of type $\sigma_{c_{(k,j,x_k^{(p)})}}$, $1 \le j \le n-1$, when its interaction environment is provided in which exists a single synapse $(c_{(k,j-1,x_k^{(p)})}, c_{(k,j,x_k^{(p)})})$ coming to the neuron. Then each rule buds a single neuron $\sigma_{c_{(k,j+1,x_k^{(p)})}}$ with a synaptic connection $(c_{(k,j,x_k^{(p)})}, c_{(k,j+1,x_k^{(p)})})$, where the second entry $(j+1)$ of the neuron label differs from the father neuron as its corresponding label entry as (j), otherwise the rest of the labeling sequence is inherited from the father neuron's label; x_k is a truth assignment of length k over $\{t, f\}$. The newly created synapse changes the interaction environment of the father neuron, which prevents another application of the rule.
As soon as the last neurons, whose second entry of the label is n, of the layer are produced, the next two types of rules are enabled to apply to those neurons as follows.

a₂) $(c_{(k,n-1,x_k^{(p)})}, c_{(k,n,x_k^{(p)})})[\ ]_{c_{(k,n,x_k^{(p)})}} \to (c_{(k,n,x_k^{(p)})}, c_{(k+1,1,t,x_k^{(p)})})[\ ]_{c_{(k+1,1,t,x_k^{(p)})}}$,
$(c_{(k,n-1,x_k^{(p)})}, c_{(k,n,x_k^{(p)})})[\ ]_{c_{(k,n,x_k^{(p)})}} \to (c_{(k,n,x_k^{(p)})}, c_{(k+1,1,f,x_k^{(p)})})[\ ]_{c_{(k+1,1,f,x_k^{(p)})}}$,
$p \in t, f$, and $1 \leq k \leq n-1$.

Those two rules of type a_2) apply simultaneously to each last neuron of type $\sigma_{c_{(k,n,x_k^{(p)})}}$ of each chain in the current layer k, the interaction environments must satisfy the rule condition. As a result, each neuron buds two new neurons with respective synapses. The next layer of the system is thus established. Hence the interaction environment of each father neuron extended by two new synapses, none of these rules is possible to apply again to those neurons. We shall look at the labels of newly produced pairs of type $\sigma_{c_{(k+1,1,t,x_k^{(p)})}}$ and $\sigma_{c_{(k+1,1,f,x_k^{(p)})}}$, the labels are formed as follows: first of all the pair $(k+1,1)$ corresponds to the neuron location where $k+1$ indicates the new layer number while 1 says the neuron is the very first one in its corresponding chain of length n in the new layer; the rest of the labeling sequence as $(t, x_k^{(p)})$ or $(f, x_k^{(p)})$ represents a new truth assignment for Boolean variables $x_1, x_2, \ldots, x_{k+1}$, where the newly inserted symbol t or f associates with a truth value $t(rue)$ or $f(alse)$, respectively, while $x_k^{(p)}$ is an heritage from the father neuron. Thus, all the possible 2^{k+1} different truth assignments are generated in layer $k+1$.

The truth assignment generation steps for two Boolean variables y_1, y_2 can be observed as described in Figure 5.

The rules of type a_1) are enabled in turn to complete the newly established layer by continued generation of the chains of length n.

By the alternated applications of the rules of types $\mathbf{a_1}$) $(n-1$ times) and $\mathbf{a_2}$) (once), in n^2 steps the layers from 0 to n are, the dendritic-tree, constructed by means an exponential work space and all the truth assignments of length n are generated.

Now, we come to the part B of the algorithm.

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 function of this part of the algorithm is to design the substructure which is devoted to the test of the satisfiability of truth assignments against the clauses and to the exploration of the possibility whether any solution to the clauses of the propositional formula exists.

The very first task in part **B** is to connect the layer n to the generator block appropriately according to the truth assignments formed in this layer. We recall here that, in layer n, there are 2^n number of subsystems each one is composed of a sequence of n neurons (chains). However, each subsystem injectively corresponds to a different truth assignment of length n.

More precisely, taking the label of a neuron $\sigma_{c_{(n,j,x_n)}}$ in layer n, where the subsequence $x_n = (x_{n1}, x_{n2}, \ldots, x_{nn}) \in \{t,f\}^n$ represents a truth assignment. We associate jth entry of x_n with the jth neuron of considering subsystem, thus,

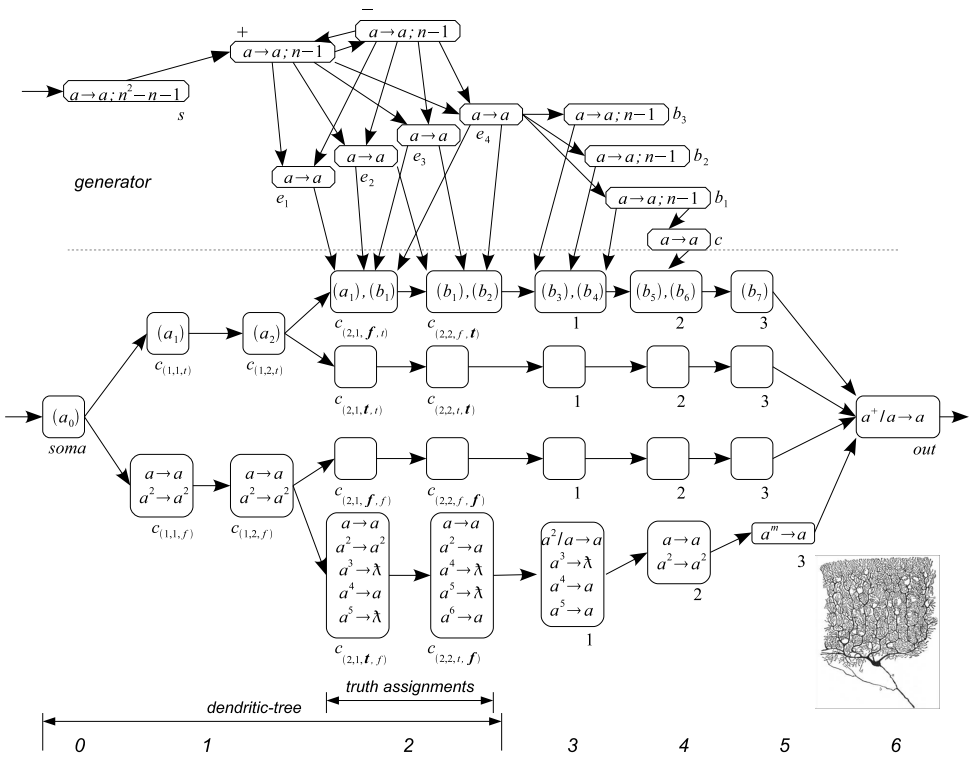

Fig. 5. An almost complete structure of Π system for SAT$(2, m)$ (maturated dendrite tree). The neuron budding rules used in each computation step are indicated by their labels in the corresponding neurons, while the spiking rules are presented too.

each neuron is indicated by an abstract triple $(n, j, x_n(j))$, where $1 \leq j \leq n$, and associated with a truth value x_{nj}. This way, a truth assignment of length n is represented by the n neurons (labels) of a subsystem.

For instance, in a case $n = 2$ as described in Figure 5, $2^2 = 4$ different truth assignments of length 2 have been generated for two Boolean variables y_1 and y_2 and each one is associated with a subsystem of layer $n = 2$. Technically, the first subsystem is composed of two neurons with labels $c_{(2,1,f,t)}$ and $c_{(2,2,f,t)}$, respectively. Whereas the former one associates with Boolean $t(rue)$ value as $x_2 = (f, t)$ and $x_2(1) = f$, while the later one with $f(alse)$ value as $x_2(2) = t$, and then altogether forms an assignment (f, t); the case with other subsystems are the same where (t, t), (f, f), (t, f), respectively, are generated; one can see that the four truth assignments are well distinguished from each other by the layer structure of four subsystems (chains).

The next synapse creation (budding) rules establish three synapses coming to the neurons which associate with a Boolean $t(rue)$ value while four synapses to the neurons associated with $f(alse)$ value, from the generator block.

b₁) $(c_{(n,j-1,x_n)}, c_{(n,j,x_n)})[\ ]_{c_{(n,j,x_n(j)=p)}} \rightarrow (c_{(n,j,x_n(j)=p)}, c_{e_i})[\ ]_{e_i}$,
$1 \leq j \leq n$, $p \in \{t, f\}$ and $s \leq i \leq 3$, where $s = 1$ if $p = t$, $s = 0$ if $p = f$, $c_{(n,0,x_n)} = c_{(n-1,n,x_n)}$.
Those neurons $\sigma_{c_{(n,j,x_n(j)=t)}}$ whose interaction environment satisfies the condition of the rules b_1) create three synapses coming from the neurons $\sigma_{c_{e_i}}$, $1 \leq i \leq 3$, while the neurons $\sigma_{c_{(n,j,x_n(j)=f)}}$ establish synapses coming to it from the four neurons $\sigma_{c_{e_i}}$, $0 \leq i \leq 3$, of the generator block. The synapse creation rules of type b_1) and the neuron budding rules of type a_1) are applied to the same neurons in layer n at the same time in a consequent $n-1$ steps as their interaction environments coincide.
Again looking at Figure 5, neuron $\sigma_{c_{(2,1,f,t)}}$ associates with $f(alse)$ value gets 4 synapses from the generator as neuron $\sigma_{c_{(2,1,f,t)}}$ gets 3 cause its identity of $t(rue)$ value.

b₂) $(c_{(n,n-1,x_n)}, c_{(n,n,x_n)})[\ ]_{c_{(n,n,x_n)}} \rightarrow (c_{(n,n,x_n)}, 1)[\ ]_1$.
The rule of type b_2) applies parallel to the last neurons of the layer n and produce the neurons σ_1 forming a new layer $n+1$. Meantime the rules of type b_1) create synapses from the same neurons of layer n to the generator block at last.

b₃) $(c_{(n,n,x_n)}, 1)[\ ]_1 \rightarrow (1, 2)[\ ]_2$,

b₄) $(c_{(n,n,x_n)}, 1)[\ ]_1 \rightarrow (b_i, 1)[\ ]_{b_i}$, $1 \leq i \leq 3$.
As rules of type b_3) apply to the neurons σ_1 and bud neurons σ_2, rules of type b_4) apply too and create three synapses coming from the neurons σ_{b_i}, $1 \leq i \leq 3$, to each neuron σ_1. Thus, layer $n+2$ is formed.

b₅) $(1, 2)[\ ]_2 \rightarrow (2, 3)[\ ]_3$,

b₆) $(1, 2)[\ ]_2 \rightarrow (c, 2)[\ ]_c$.
The rules of types b_5) and b_6) apply simultaneously to a neuron σ_2 with a synapse (1,2). As a result, the former one buds a new neuron σ_3, while the later one makes a new connection coming from the neuron σ_c as $(c, 2)$. All other neurons σ_2 get the same effect by the rules as the maximal parallel applications of the rules.

b₇) $(2, 3)[\ ]_3 \rightarrow (3, out)[\ ]_{out}$.
The pre-computation of the SN P system structure construction is completed by forming the converged connections from the neurons σ_3 to the output neuron σ_{out}, by means the rules of type b_7) are applied parallel to all neurons of layer $n+3$.

Thus, the SN P system device structure totally empty of spikes which is to solve all the instances of SAT(n, m), has been (pre-)computed in a polynomial time. The next computation stage (post-computation) to solve SAT(n, m) is plugged-in as follows.

Solving SAT

Any given instance γ_n of SAT(n, m) is encoded in a sequence of spikes. Each clause C_i of γ_n can be seen as a disjunction of at most n literals: for each $j \in \{1, 2, \ldots, m\}$, either y_j occurs in C_i, or $\neg y_j$ 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:

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

In this way, 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 $\gamma_3 = (y_1 \vee \neg y_2) \wedge (y_1 \vee y_3)$ is $aa^2\lambda a\lambda a$.

The spiking rules residing in the neurons of the system which perform for solving the introduced problem are listed below with a brief description for each. But we do not go detailed explanation of each rule functions here, we prefer it refer to Section 3 and the paper [6], also the neuron budding rules are out of usage in this stage.

A given instance $\gamma_n \in$ SAT(n, m) encoded in a spike sequence is introduced into the system structure and will be processed by the spiking rules according to their roles in each step of the computation.

c₁) $[a \to a]_{c_{soma}}$; $[a^2 \to a^2]_{c_{soma}}$;
$[a \to a; n^2 - n - 1]_s$.
At each computation step of introducing the input, we insert 0, 1 or 2 spikes into the system through the input neuron σ_{soma}, according to the value of the spike variable α_{ij} we are considering in the representation of γ_n. Meantime we insert a single spike a into neuron σ_s once, which excites the generator block.

c₂) $[a \to a]_{c_{(k,j,x_k)}}$; $[a^2 \to a^2]_{c_{(k,j,x_k)}}$
$1 \leq k \leq n-1, 1 \leq j \leq n, x_k \in \{t, f\}^k$.
Each spike inserted into the input neuron is duplicated here and transmit along the first layer of the system towards next layers. When a spike passes a touching point – neuron with label of type $c_{(k,n,x_k)}$, it is duplicated and enter into next layer, etc., finally 2^n copies of them will take place at the layer n.

Once the copies of a clause are taken place on the neurons of the chains of length n in layer n, the combined functioning of the generator block and the layer n tests the assignments against each copy of the clause in consideration. For this purpose, the rules **c₃)** $-$ **c₅)** are used.

c₃) $[a \to a]_{e_i}; 0 \leq i \leq 3$,
$[a \to a; n-1]_+$; $[a \to a; n-1]_-$.

The generator block and its spiking rules. The generator block provides 3 and 4 spikes, respectively, in each n steps to the neurons associated with truth values t and f, of layer n, in order to test the satisfiability of the truth assignments against a clause which has been taken place through the layer.

c_4) $[a \to a]_{t_t}$; $[a^3 \to \lambda]_{t_t}$; $[a^2 \to a^2]_{t_1}$;
$[a^4 \to a]_{t_t}$; $[a^5 \to \lambda]_{t_t}$; $[a^2 \to a]_{t_0}$;
$t_t = c_{(n,j,x_n(j)=t)}, 1 \leq j \leq n$,
$t_1 = c_{(n,j,x_n(j)=t)}, 1 \leq j \leq n-1$,
$t_0 = c_{(n,n,x_n(n)=t)}, x_n \in \{t,f\}^n$.
The spiking rules residing in the neurons which associate Boolean $t(rue)$ value in layer n. The rules $a^2 \to a^2$, $a^2 \to a$, and $a \to a$ used to transmit the spike variables a, a^2 along the chains. Once a clause placed, each neuron associated with $t(rue)$ value contains either of a spike a or a^2 or *empty*. As a spike variable a represents a truth variable y, to which a spike *true* value a^3 sent by the generator is assigned and it results an *yes* answer as a^4, then it passes to the neuron σ_1 along the chain with a saying that a truth variable of the clause is satisfied by true value of a truth assignment or simply the clause is satisfied by a truth assignment of the corresponding chain. Meanwhile, a true value a^3 is assigned to the spike variables a^2 associates to truth variable $\neg y$ and *empty* wherever, which give a result *no* by means the rules $a^3 \to \lambda$ and $a^5 \to \lambda$ are performed.

c_5) $[a \to a]_{f_f}$; $[a^4 \to \lambda]_{f_f}$; $[a^2 \to a^2]_{f_1}$;
$[a^5 \to \lambda]_{f_f}$; $[a^6 \to a]_{f_f}$; $[a^2 \to a]_{f_0}$;
$f_f = c_{(n,j,x_n(j)=f)}, 1 \leq j \leq n$,
$f_1 = c_{(n,j,x_n(j)=f)}, 1 \leq j \leq n-1$,
$f_0 = c_{(n,n,x_n(n)=f)}, x_n \in \{t,f\}^n$.
The spiking rules residing in the neurons which associate with Boolean $f(alse)$ value in layer n. The functioning of the rules is similar as rules c_5).

c_6) $[a \to a; n-1]_{b_i}; 1 \leq i \leq 3$,
$[a^2/a \to a]_1$; $[a^3 \to \lambda]_1$;
$[a^4 \to a]_1$; $[a^5 \to a]_1$.
Whether an assignment satisfies the considered clause or not is checked by a combined functioning of the neurons with label 1 in layer $n+1$ and the neurons with label $b_i, 1 \leq i \leq 3$.

c_7) $[a \to \lambda]_2$; $[a^2 \to a]_2$;
$[a \to a]_c$.
With a combined function of neuron σ_c, neuron σ_2 emits a spike into neuron σ_3 if the corresponding assignment satisfies the under consideration clause here, otherwise no spike is emitted.

c_8) $[a^m \to a]_3$;
$[a^+/a \to a]_{out}$.
Neurons with label 3 count how many clauses of the instance γ_n are satisfied by the associated truth assignments. If any of those neurons get m spikes, which fire, hence the number of spikes that reach neuron *out* is the number of

assignments that satisfy all the clauses of γ_n. Thus, the output neuron fires if it has got at least one spike by means the problem has a positive solution, otherwise there is no assignment satisfies the instance γ_n.

This stage of the computation ends at the $n^2 + nm + 4$th step. The entire system halts in total at most in $2(n^2 + nm + 4)$ number of computation steps.

Thus, we got a full (deterministic, polynomial time and uniform) solution to SAT(n,m) in the framework of SN P systems.

5 Discussion

The present paper concerns the efficiency of SN P systems, we proposed a way to solve **NP**-complete problems, particularly SAT, in polynomial time. Specifically, the *neuron budding rule* is introduced in the framework of SN P systems, which a new feature enhances the efficiency of the systems to generate necessary work space. Neuron budding rules drive the mechanism of neuron production and synapse creation according to the interaction of a neuron with its environment (described by its synapses connected to other neurons). A very restricted type of rule of neuron budding, at most one synapse is involved in an environment, is used, but it is powerful enough to solve the considered problem, SAT. The solution to SAT by SN P systems with neuron budding contains two computation stages: first, constructing the device structure which has no spikes inside, second, introducing the considered problem to be solved encoded in spikes into the device. The system works in deterministic and maximally parallel manner. The whole mechanism we considered here for solutions to computational intractable problems is elegant from computational complexity theory point of view as the designed algorithm can be computed by a deterministic Turing machine in polynomial time; the operation of neuron budding is well motivated by neural biology.

We believe that SN P systems use the restricted budding rules can be an efficient computing tool to solve other **NP** hard problems.

The 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 environment. One possible form of such general rules is as follows: $A_i[\ ]_i B_i \rightarrow C_j[\ ]_j D_j$, where A_i, B_i and C_j, D_j are the set 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 are involved in the environment of the considered neuron.

Acknowledgments

The work of Tseren-Onolt Ishdorj was supported by the project "Biotarget", it is a joint project between U. of Turku and Åbo Akademi University funded by the Academy of Finland. The work of L. Pan was supported by 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 Natural Science Foundation of Hubei Province (2008CDB113 and 2008CDB180).

References

1. H. Chen, M. Ionescu, T.-O. Ishdorj: On the efficiency of spiking neural P systems. *Proc. 8th Inter. Conf. on Electronics, Information, and Communication*, Ulanbator, Mongolia, June 2006, 49–52.
2. 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, 241–266.
3. H. Chen, R. Freund, M. Ionescu, Gh. Păun, M.J. Pérez-Jiménez: On string languages generated by spiking neural P systems. *Fundamenta Informaticae* 75 (2007), 141–162.
4. V. Danos, J. Feret, W. Fontana, R.Harmer and J. Krivine, *Investigation of a Biological Repair Scheme*, In Proceedings of 9th International Workshop on Membrane Computing, Edinburgh, UK, July 28-31, 2008, LNCS 5391, 1–12, 2009.
5. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71, 2–3 (2006), 279–308.
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 (2008), 519–534.
7. T.-O. Ishdorj, A. Leporati, L. Pan, X. Zhang, X. Zeng: Uniform solutions to `QSAT` and `Q3SAT` by spiking neural P systems with pre-computed resources. In the present volume.
8. M.A. Gutiérrez-Naranjo, A. Leporati: Solving numerical NP-complete problems by spiking neural P systems with pre-computed resources. *Proceedings of Sixth Brainstorming Weeks on Membrane Computing*, Sevilla University, Sevilla, February 2-8, 2008, 193–210.
9. A. Leporati, C. Zandron, C. Ferretti, G. Mauri: Solving numerical NP-complete problem with spiking neural P systems. *Membrane Computing, International Workshop, WMC8, Thessaloniki, Greece, 2007, Selected and Invited Papers* (G. Eleftherakis, P. Kefalas, Gh. Păun, G. Rozenberg, A. Salomaa, eds.), LNCS 4860, Springer-Verlag, Berlin, 2007, 336–352.
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. Submitted.
11. A. Leporati, C. Zandron, C. Ferretti, G. Mauri: On the computational power of spiking neural P systems. *International Journal of Unconventional Computing*, 2007, in press.
12. Gh. Păun: *Membrane Computing – An Introduction*. Springer-Verlag, Berlin, 2002.
13. Gh. Păun: Twenty six research topics about spiking neural P systems. *Fifth brainstorming week on membrane computing*, Fenix Editora, Sevilla, 2007, 263-280.
14. M. Sipser: *Introduction to the Theory of Computation*. PWS Publishing Company, Boston, 1997.
15. Think and Grow Toys: `http://www.tagtoys.com/dendrites.htm`