Chapter 4

Spiking Neural P Systems with Neuron Division

Abstract
Spiking neural P systems (SN P systems, for short) are a class of distributed parallel computing devices inspired from the way neurons communicate by means of spikes. The features of neuron division and neuron budding are recently introduced into the framework of SN P systems, and it was shown that SN P systems with neuron division and neuron budding can efficiently solve computationally hard problems. In this work, the computation power of SN P systems with neuron division only, without budding, is investigated; it is proved that a uniform family of SN P systems with neuron division can efficiently solve SAT in a deterministic way, not using budding, while additionally limiting the initial size of the system to a constant number of neurons. This answers an open problem formulated by Pan et al.

4.1 Introduction

Spiking neural P systems (SN P systems, for short) have been introduced in [16] 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 (see, e.g., [12, 24, 25]). The resulting models are a variant of tissue-like and neural-like P systems from membrane computing. Please refer to the classic [30] for the basic information about membrane computing, to the handbook [35] for a comprehensive presentation, and to the web site [1] for the up-to-date information.

In short, an SN P system consists of a set of neurons placed in the nodes of a directed graph, which send signals (spikes) along synapses (arcs of the graph). Each neuron contains a number of spikes, and is associated with a number of firing and forgetting rules: within the system the spikes are moved, created, or
deleted.

The computational efficiency of SN P systems has been recently investigated in a series of works [5, 18, 19, 20, 21, 22, 23]. An important issue is that of uniform solutions to NP-complete problems, i.e., where the construction of the system depends on the problem and not directly on the specific problem instance (if may, however, depend on the size of the instance). Within this context, most of the solutions exploit the power of nondeterminism [21, 22, 23] or use pre-computed resources of exponential size [5, 18, 19, 20].

Recently, another idea is introduced for constructing SN P systems to solve computationally hard problems by using neuron division and budding [21], where for all \( n, m \in \mathbb{N} \) all the instances of \( SAT(n, m) \) with at most \( n \) variables and at most \( m \) clauses are solved in a deterministic way in polynomial time using a polynomial number of initial neurons. As both neuron division rules and neuron budding rules are used to solve \( SAT(n, m) \) problem in [21], it is a natural question to design efficient SN P systems omitting either neuron division rules or neuron budding rules for solving NP-complete problem.

In this work, a uniform family of SN P systems with only neuron division is constructed for efficiently solving SAT problem, which answers the above question posed in [21]. Additionally, the result of [21] is improved in the sense that the SN P systems are constructed with a constant number of initial neurons instead of linear number with respect to the parameter \( n \), while the computations still last a polynomial number of steps.

The paper is organized as follows. In the next section the definition of SN P systems with neuron division rules is given. In Section 4.3 a uniform family of SN P systems is constructed with a constant number of initial neurons, which can solve SAT problem in a polynomial time. Conclusions and remarks are given in Section 4.4.

### 4.2 SN P Systems with Neuron Division

Readers are assumed to be familiar with basic elements about SN P systems, e.g., from [16] and [1], and formal language theory, as available in many monographs. Here, only SN P systems with neuron division are introduced.

A spiking neural P system with neuron division is a construct \( \Pi \) of the following form:

\[
\Pi = (\{a\}, H, \text{syn}, n_1, \ldots, n_m, R, \text{in}, \text{out}),
\]

where:

1. \( m \geq 1 \) (the initial degree of the system);
2. \( a \) is an object, called spike;
3. \( H \) is a finite set of labels for neurons;
4. syn ⊆ H × H is a synapse dictionary between neurons; with (i, i) \notin \text{syn} for i \in H;

5. n_i \geq 0 is the initial number of spikes contained in neuron i, i \in \{1, 2, \ldots, m\};

6. R is a finite set of developmental rules, of the following forms:

   (1) extended firing (also 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 division rule \([E]_i \rightarrow [ ]_j \parallel [ ]_k\), where E is a regular expression over a and i, j, k \in H;

7. in, out \in H indicate the input and the output neurons of \Pi.

Several shorthand notations are customary for SN P systems. If a rule \([E/a^c \rightarrow a^p; d]_i\) has \(E = a^c\), then it is written in the simplified form \([a^c \rightarrow a^p; d]_i\); similarly, if it has \(d = 0\), then it is written as \([E/a^c \rightarrow a^p]_i\); of course notation for \(E = a^c\) and \(d = 0\) can be combined into \([a^c \rightarrow a^p]_i\). A rule with \(p = 0\) is called extended forgetting rule.

If a neuron \(\sigma_i\) (a notation used to indicate it has label i) contains \(k\) spikes and \(a^k \in L(E)\), \(k \geq c\), where \(L(E)\) denotes the language associated with the regular expression \(E\), then the rule \([E/a^c \rightarrow a^p; d]_i\) is enabled and it can be applied. It means that \(c\) spikes are consumed, \(k - c\) spikes remain in the neuron, and \(p\) spikes are produced after \(d\) time units. If \(d = 0\), then the spikes are emitted immediately; if \(d \geq 1\) and the rule is used in step \(t\), then in steps \(t, t + 1, t + 2, \ldots, t + d - 1\) the neuron is closed and it cannot receive new spikes (these particular input spikes are “lost”, that is, they are removed from the system). In the step \(t + d\), the neuron spikes and becomes open again, so that it can receive spikes. Once emitted from neuron \(\sigma_i\), the \(p\) spikes reach immediately all neurons \(\sigma_j\) such that there is a synapse going from \(\sigma_i\) to \(\sigma_j, i.e., (\sigma_i, \sigma_j) \in \text{syn},\) and which are open. Of course, if neuron \(\sigma_i\) has no synapse leaving from it, then the produced spikes are lost.

If the rule is a forgetting one of the form \([E/a^c \rightarrow \lambda]_i\), then, when it is applied, \(c \geq 1\) spikes are removed, but none are emitted.

If a neuron \(\sigma_i\) contains \(s\) spikes and \(a^s \in L(E)\), then the division rule \([E]_i \rightarrow [ ]_j \parallel [ ]_k\) can be applied, consuming \(s\) spikes the neuron \(\sigma_i\) is divided into two neurons, \(\sigma_j\) and \(\sigma_k\). The child neurons contain no spike in the moment when they are created, but they contain developmental rules from \(R\) and inherit the synapses that the parent neuron already has, i.e., if there is a synapse from neuron \(\sigma_g\) to the parent neuron \(\sigma_i\), then in the process of division, one synapse from neuron \(\sigma_g\) to child neuron \(\sigma_j\) and another one from \(\sigma_g\) to \(\sigma_k\) are established. The same holds when the connections are in the other direction (from the parent neuron \(\sigma_i\) to a neuron \(\sigma_g\)). In addition to the inheritance of synapses, the child neurons can have new synapses as provided by the synapse dictionary. If a child neuron \(\sigma_g, g \in \{j, k\}\), and another neuron \(\sigma_h\) have the relation \((g, h) \in \text{syn}\) or
In each time unit, for each neuron, if a neuron can use one of its rules, then a rule from \( R \) must be used. For the general model, if several rules are enabled in the same neuron, then only one of them is chosen non-deterministically. In this paper however all neurons behave deterministically and there will be no conflict between rules. When a neuron division rule is applied, at this step the associated neuron is closed, it cannot receive spikes. In the next step, the neurons obtained by division will be open and can receive spikes. Thus, the rules are used in the sequential manner in each neuron, but neurons function in parallel with each other.

The configuration of the system is described by the topological 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.

If \( m \) is the initial degree of the system, then the initial configuration of the system consists of neurons \( \sigma_1, \ldots, \sigma_m \) with labels 1, \ldots, \( m \) and connections as specified by the synapse dictionary \( \text{syn} \) for these labels. Initially \( \sigma_1, \ldots, \sigma_m \) contain \( n_1, \ldots, n_m \) spikes (respectively).

In the next section, the input of a system is provided by a sequence of several spikes entering the system in a number of consecutive steps via the input neuron. Such a sequence is written in the form \( a^{i_1}.a^{i_2}.\cdots.a^{i_r} \), where \( r \geq 1, i_j \geq 0 \) for each \( 1 \leq j \leq r \), which means that \( i_j \) spikes are introduced in neuron \( \sigma_{in} \) in step \( j \) of the computation.

### 4.3 Solving SAT

In this section, a uniform family of SNP systems with neuron division is constructed for efficiently solving SAT, the most invoked NP-complete problem [11]. The instances of SAT consist of two parameters: the number \( n \) of variables and a propositional formula which is a conjunction of \( m \) clauses, \( \gamma = C_1 \land C_2 \land \cdots \land C_m \). Each clause is a disjunction of literals, occurrences of \( x_i \) or \( \neg x_i \), built on the set \( X = \{ x_1, x_2, \ldots, x_n \} \) of variables. An assignment of the variables is a mapping \( p : X \rightarrow \{0,1\} \) that associates to each variable a truth value. We say that an assignment \( p \) satisfies the formula \( \gamma \) if, once the truth values are assigned to all the variables according to \( p \), the evaluation of \( \gamma \) gives 1 (true) as a result (meaning that in each clause at least one of the literals must be true).

The set of all instances of SAT with \( n \) variables and \( m \) clauses is denoted by \( SAT(n,m) \). Because the construction is uniform, any given instance \( \gamma \) of \( SAT(n,m) \) needs to be encoded. Here, the way of encoding given in [21] is followed. As each clause \( C_i \) of \( \gamma \) is a disjunction of at most \( n \) literals, and thus for each \( j \in \{1,2,\ldots,n\} \) either \( x_j \) occurs in \( C_i \), or \( \neg x_j \) occurs, or none of them occurs. In order to distinguish these three situations the spike variables \( \alpha_{i,j} \) are
defined, for $1 \leq i \leq m$ and $1 \leq j \leq n$, whose values are amounts of spikes assigned as follows:

$$\alpha_{i,j} = \begin{cases} 
    a & \text{if } x_j \text{ occurs in } C_i, \\
    a^2 & \text{if } \neg x_j \text{ occurs in } C_i, \\
    a^0 & \text{otherwise.}
\end{cases}$$

In this way, clause $C_i$ will be represented by the sequence $\alpha_{i,1}\alpha_{i,2}\ldots\alpha_{i,n}$ of spike variables. In order to give the systems enough time to generate the necessary workspace before computing the instances of SAT$(n,m)$, a spiking train $(a^0)^4$ is added in front of the formula encoding spike train. Thus, for any given instance $\gamma$ of SAT$(n,m)$, the encoding sequence equals $\text{cod}(\gamma) = (a^0)^4\alpha_{1,1}\alpha_{1,2}\ldots\alpha_{1,n}\alpha_{2,1}\alpha_{2,2}\ldots\alpha_{2,n}\ldots\alpha_{m,1}\alpha_{m,2}\ldots\alpha_{m,n}$.

For each $n, m \in \mathbb{N}$, a system of initial degree 11 is constructed,

$$\Pi((n,m)) = \{\{a\}, H, \text{syn, } n_1, \ldots, n_{11}, R, \text{in, out}\},$$

with the following components:

\[
H = \{\text{in, out}\} \cup \{0, 1, 2, 3, 4\} \\
    \cup \{b_i \mid i = 1, 2, \ldots, n - 1\} \cup \{d_i \mid i = 0, 1, \ldots, n\} \\
    \cup \{e_i \mid i = 1, 2, \ldots, n - 1\} \cup \{Cx_i \mid i = 1, 2, \ldots, n\} \\
    \cup \{g_i \mid i = 1, 2, \ldots, n - 1\} \cup \{h_i \mid i = 1, 2, \ldots, n\} \\
    \cup \{Cx_{i0} \mid i = 1, 2, \ldots, n\} \cup \{Cx_i1 \mid i = 1, 2, \ldots, n\} \\
    \cup \{t_i \mid i = 1, 2, \ldots, n + 1\} \cup \{f_i \mid i = 1, 2, \ldots, n + 1\};
\]

\[
syn = \{(1, b_1), (1, e_1), (1, g_1), (1, 2), (3, 4), (4, 0), (0, \text{out})\} \\
    \cup \{(i + 1, i) \mid i = 0, 1, 2\} \cup \{(d_n, d_1), (d_n, 4)\} \\
    \cup \{(d_i, d_{i+1}) \mid i = 0, 1, \ldots, n - 1\} \cup \{(\text{in, } Cx_i) \mid i = 1, 2, \ldots, n\} \\
    \cup \{(d_i, Cx_i) \mid i = 1, 2, \ldots, n\} \cup \{(Cx_i, h_i) \mid i = 1, 2, \ldots, n\} \\
    \cup \{(Cx_{i1}, t_i) \mid i = 1, 2, \ldots, n\} \cup \{(Cx_{i0}, f_i) \mid i = 1, 2, \ldots, n\};
\]

labels of the initial neurons are: in, out, $d_0, b_1, e_1, g_1, 0, 1, 2, 3, 4$; the initial contents are $n_{d_0} = 5$, $n_{b_1} = n_{e_1} = n_{g_1} = n_2 = 2$, $n_3 = 7$, and there is no spike in the other neurons;

$R$ is the following set of rules:

(A) rules for the ‘Generation stage’:

\[
\begin{align*}
    [a^2]_{h_i} & \rightarrow [\varepsilon_i], \varepsilon_i = 1, 2, \ldots, n - 1, \\
    [a^2]_{e_{i-1}} & \rightarrow [\varepsilon_{i-1}], \varepsilon_{i-1} = 1, 2, \ldots, n - 1, \\
    [a^2]_{e_{i-1}} & \rightarrow [\varepsilon_i], \varepsilon_i = 1, 2, \ldots, n - 1, \\
    [a^2]_{h_{n-1}} & \rightarrow [\varepsilon_n], \varepsilon_n = 1, 2, \ldots, n - 1, \\
    [a^2]_{g_{i+1}} & \rightarrow [\varepsilon_{i+1}], \varepsilon_{i+1} = 1, 2, \ldots, n - 1, \\
    [a^2]_{g_i} & \rightarrow [\varepsilon_i], \varepsilon_i = 1, 2, \ldots, n - 1, \\
    [a^2]_{h_{n-1}} & \rightarrow [\varepsilon_n], \varepsilon_n = 1, 2, \ldots, n - 1, \\
    [a^2]_{h_i} & \rightarrow [\varepsilon_i], \varepsilon_i = 1, 2, \ldots, n, \\
    [a \rightarrow \lambda]_{d_i} & \rightarrow [\varepsilon_i], \varepsilon_i = 1, 2, \ldots, n,
\end{align*}
\]
Solving \( SAT \)

\[
[a^2 \rightarrow \lambda]_{d_i}, \; i = 1, 2, \ldots, n,
\]
\[
a \rightarrow \lambda]_{C_{x_i}, \; i = 1, 2, \ldots, n},
\]
\[
[a^2 \rightarrow \lambda]_{C_{x_i}, \; i = 1, 2, \ldots, n},
\]
\[
a \rightarrow \lambda]_{C_{x_1}, \; i = 1, 2, \ldots, n},
\]
\[
[a^2 \rightarrow \lambda]_{C_{x_1}, \; i = 1, 2, \ldots, n},
\]
\[
a \rightarrow \lambda]_{C_{x_0}, \; i = 1, 2, \ldots, n},
\]
\[
[a^2 \rightarrow \lambda]_{C_{x_0}, \; i = 1, 2, \ldots, n},
\]
\[
a \rightarrow a]_{i}, \; i = 1, 2,
\]
\[
[a^2 \rightarrow a^2]_{i}, \; i = 1, 2,
\]
\[
a^3 \rightarrow \lambda]_{2},
\]
\[
a^4 \rightarrow a]_{2},
\]
\[
a^7/a^2 \rightarrow a^2; 2n - 3]_3,
\]
\[
a^5/a^2 \rightarrow a^2; 2n - 1]_3,
\]
\[
a^2 \rightarrow \lambda]_{4},
\]
\[
a^2 \rightarrow \lambda]_{0},
\]
\[
a]_0 \rightarrow [ ]_{t_1} \parallel [ ]_{f_1},
\]
\[
a]_{i} \rightarrow [ ]_{t_{i+1}} \parallel [ ]_{f_{i+1}}, \; i = 1, 2, \ldots, n - 1,
\]
\[
[a]_{f_{i}} \rightarrow [ ]_{t_{i+1}} \parallel [ ]_{f_{i+1}}, \; i = 1, 2, \ldots, n - 1;
\]

(B) rules for the ‘Input stage’:

\[
a \rightarrow a]_{in},
\]
\[
a^2 \rightarrow a^2]_{in},
\]
\[
a^4/a^3 \rightarrow a^3; 4n]_{d_0},
\]
\[
a \rightarrow a; nm - 1]_{d_0},
\]
\[
a^3 \rightarrow a^3]_{d_i}, \; i = 1, 2, \ldots, n,
\]
\[
a^4 \rightarrow \lambda]_{d_i},
\]
\[
a^3 \rightarrow \lambda]_{C_{x_i}, \; i = 1, 2, \ldots, n},
\]
\[
a^4 \rightarrow \lambda]_{C_{x_i}, \; i = 1, 2, \ldots, n},
\]
\[
a^5 \rightarrow \lambda]_{C_{x_0}, \; i = 1, 2, \ldots, n},
\]
\[
a^5 \rightarrow \lambda]_{C_{x_0}, \; i = 1, 2, \ldots, n},
\]
\[
a^3 \rightarrow a^3; nm + 2]_3,
\]
\[
a^3 \rightarrow a; 1]_4,
\]
\[
a^6 \rightarrow a^2; 1]_4,
\]
\[
a]_{t_{n}} \rightarrow [ ]_{t_{n+1}} \parallel [ ]_{f_{n+1}};
\]
\[
a^{3k+1} \rightarrow \lambda]_{t_{i}}, \; 1 \leq k \leq n, \; i = 1, 2, \ldots, n,
\]
\[
a^{3k+2}/a^2 \rightarrow a^2]_{t_{i}}, \; 1 \leq k \leq n, \; i = 1, 2, \ldots, n,
\]
\[
a]_{f_{n}} \rightarrow [ ]_{t_{n+1}} \parallel [ ]_{f_{n+1}},
\]
\[
a^{3k+1} \rightarrow \lambda]_{f_{i}}, \; 1 \leq k \leq n, \; i = 1, 2, \ldots, n,
\]
\[
a^{3k+2}/a^2 \rightarrow a^2]_{f_{i}}, \; 1 \leq k \leq n, \; i = 1, 2, \ldots, n,
\]
\[
a^{3k+1} \rightarrow \lambda]_{t_{n+1}, \; 0 \leq k \leq n},
\]
\[
a^{3k+2} \rightarrow \lambda]_{t_{n+1}, \; 0 \leq k \leq n}.
\]

(C) rules for the ‘Satisfiability checking stage’:
To solve the \textsc{sat} problem in the framework of SN P systems with neuron division, the strategy consists of four phases, as in [21]: \textit{Generation Stage}, \textit{Input Stage}, \textit{Satisfiability Checking Stage} and \textit{Output Stage}. In the first stage, the neuron division is applied to generate necessary neurons to constitute the input and satisfiability checking modules, i.e., each possible assignment of variables $x_1, x_2, \ldots, x_n$ is represented by a neuron (with associated connections with other neurons by synapses). In the input stage, the system reads the encoding of the given instance of \textsc{sat}. In the satisfiability checking stage, the system checks whether or not there exists an assignment of variables $x_1, x_2, \ldots, x_n$ that satisfies all the clauses in the propositional formula $C$. In the last stage, the system sends a spike to the environment only if the answer is positive; no spikes are emitted in case of a negative answer.

The initial structure of the original system from [21] is shown in Figure 1, where the initial number of neurons is $4n + 7$ and an exponential number of neurons are generated by the neuron division and budding rules. In this work, the initial number of neurons is reduced to constant 11 and only the neuron division rule is used to generate the SN P systems. The division and budding rules are not indicated in Figure 1: the process starts with neuron $\sigma_0$ (to the right, before the output neuron), which then results in an exponential number of neurons (with a linear number of fresh labels).

Let us have an overview of the computation. In the initial structure of the system which is shown in Figure 2, there are 11 neurons: the left two neurons $\sigma_{i_0}$ and $\sigma_{i_n}$ are the first layer of the input module; the two neurons $\sigma_{b_{i_1}}$ and $\sigma_{e_{i_1}}$ and their offspring will be used to generate the second and third layers by neuron division rules respectively; the neuron $\sigma_{g_1}$ and its offspring will be used to generate the first layer of the satisfiability checking module, while $\sigma_0$ and its offspring will be used to produce an exponential workspace (the second layer of satisfiability checking module); the auxiliary neurons $\sigma_1$, $\sigma_2$ and $\sigma_3$ supply necessary spikes to the neurons $\sigma_{b_1}$, $\sigma_{e_1}$, $\sigma_{g_1}$ and $\sigma_0$ and their offspring for neuron division rules; neuron $\sigma_4$ supplies spikes to the exponential workspace in the satisfiability checking process; the neuron $\sigma_{out}$ is used to output the result.

By the encoding of instances, it is easy to see that neuron $\sigma_{i_n}$ takes $4n$ steps to read $(a^0)^{4n}$ of $\text{cod}(\gamma)$, then the spike variables $\alpha_{i,j}$ will be introduced into the neuron from step $4n + 1$. In the first $2n - 1$ steps, the system generates the second and third layers of the input module, and also the first layer of the satisfiability checking module; then in the next $2n + 1$ steps, neurons $\sigma_0$ and its offspring will be used to generate the second layer of satisfiability checking module. After that, the system reads the part of the encoding (the spike variables $\alpha_{i,j}$), checks the satisfiability and outputs the result.

\[
\begin{align*}
[a^{3k+1} \rightarrow \lambda]_{f_{n+1}}, & \quad 0 \leq k \leq n; \\
[a^{3k+2} \rightarrow \lambda]_{f_{n+1}}, & \quad 0 \leq k \leq n; \\
(D) \text{ rules for the ‘Output stage’:} & \\
[(a^2)^+ / a \rightarrow a]_{out}.
\end{align*}
\]
Figure 4.1: The initial structure the SN P system $\Pi_2$ from [21]

Figure 4.2: The initial structure of system $\Pi(\langle n, m \rangle)$
Chapter 4

Generation Stage: Neuron $\sigma_{d_0}$ initially contains 5 spikes and the rule $[a^5/a^4 \rightarrow a^4; 4n]_{d_0}$ is applied. It will emit 4 spikes at step $4n + 1$ because of the delay $4n$. In the beginning, neurons $\sigma_{b_1}$, $\sigma_{e_1}$ and $\sigma_{g_1}$ have 2 spikes respectively, their division rules are applied, and six neurons $\sigma_{d_1}$, $\sigma_{b_2}$, $\sigma_{C_{x_1}}$, $\sigma_{e_2}$, $\sigma_{h_1}$ and $\sigma_{g_2}$ are generated. They have six associated synapses $(1, d_1)$, $(1, b_2)$, $(1, C_{x_1})$, $(1, e_2)$, $(1, h_1)$ and $(1, g_2)$, where they are obtained by the heritage of synapses $(1, b_1)$, $(1, e_1)$ and $(1, g_1)$, respectively, and three new synapses $(d_0, d_1)$, $(d_1, C_{x_1})$ and $(C_{x_1}, h_1)$ are generated by the synapse dictionary. At step 1, the auxiliary neuron $\sigma_2$ sends 2 spikes to neuron $\sigma_1$, then in the next step $\sigma_1$ will send 2 spikes to neurons $\sigma_{b_2}$, $\sigma_{e_2}$, $\sigma_{h_1}$ and $\sigma_{g_2}$ for the next division. Note that neuron $\sigma_3$ has 7 spikes in the beginning, and will send 2 spikes to neuron $\sigma_2$ at step $2n - 2$ because of the delay $2n - 3$ (as we will see, at step $2n - 2$, neuron $\sigma_2$ also receive 2 spikes from neuron $\sigma_1$ and the rule $[a^4 \rightarrow a]_2$ will be applied. Hence, after step $2n - 1$, neurons $\sigma_0$ and its offspring will be used to generate an exponential workspace). The structure of the system after step 1 is shown in Figure 3.

![Figure 3: The structure of system $\Pi((n, m))$ after step 1](image)

At step 2, neuron $\sigma_1$ sends 2 spikes to neurons $\sigma_{b_2}$, $\sigma_{e_2}$, $\sigma_{h_1}$, $\sigma_{g_2}$, $\sigma_{2}$, $\sigma_{0}$, $\sigma_{d_1}$ and $\sigma_{C_{x_1}}$, respectively. In the next step, the former four neurons consume their spikes for neuron division rules; neuron $\sigma_2$ sends 2 spikes back to $\sigma_1$ (in this way, the auxiliary neurons $\sigma_1$, $\sigma_2$, $\sigma_3$ supply 2 spikes for division every two steps in the first $2n - 2$ steps); the spikes in the other three neurons $\sigma_0$, $\sigma_{d_1}$, and $\sigma_{C_{x_1}}$ are deleted by the rules $[a^2 \rightarrow \lambda]_{d_1}$, $[a^2 \rightarrow \lambda]_{C_{x_1}}$ and $[a^2 \rightarrow \lambda]_{0}$, respectively. At step 3, neurons $\sigma_{b_2}$, $\sigma_{e_2}$, $\sigma_{h_1}$ and $\sigma_{g_2}$ are divided, eight new neurons $\sigma_{d_2}$, $\sigma_{b_3}$, $\sigma_{C_{x_2}}$, $\sigma_{e_3}$, $\sigma_{C_{x_1}}$, $\sigma_{C_{x_3}}$, $\sigma_{h_2}$ and $\sigma_{g_3}$ are generated, and the associated synapses are obtained by heritage or synapse dictionary. The corresponding structure of the system after step 3 is shown in Figure 4.

The neuron division is iterated until neurons $\sigma_{d_i}$, $\sigma_{C_{x_1}}$, $\sigma_{C_{x_1}}$ and $\sigma_{C_{x_3}}$ (1 ≤ $i$ ≤ $n$) are obtained at step $2n - 1$. Note that the division rules in neurons $\sigma_{b_{n-1}}$, $\sigma_{e_{n-1}}$ and $\sigma_{g_{n-1}}$ are slightly different with those division rules in neurons $\sigma_{b_i}$, $\sigma_{e_i}$ and $\sigma_{g_i}$ (1 ≤ $i$ ≤ $n - 2$). At step $2n - 2$, neuron $\sigma_3$ sends 2 spikes to neuron $\sigma_2$. At
the same time, neuron $\sigma_1$ also sends 2 spikes to $\sigma_2$. So neuron $\sigma_2$ sends one spike to $\sigma_1$ by the rule $[a^4 \rightarrow a]_2$ is applied at step $2n - 1$. Similarly, the auxiliary neurons $\sigma_1$, $\sigma_2$, $\sigma_3$ supply one spike every two steps to generate an exponential workspace from step $2n - 1$ to $4n$ (neuron $\sigma_0$ and its offspring use the spikes for division, while neurons $\sigma_d$, $\sigma_{Cx_1}$, $\sigma_{Cx_2}$ and $\sigma_{Cx_0}$ delete the spikes received). Note that the synapses $(d_n, d_1)$ and $(d_n, 4)$ are established by the synapse dictionary. The structure of the system after step $2n - 1$ is shown in Figure 5.

At step $2n$, neuron $\sigma_0$ has one spike coming from $\sigma_1$, the rule $[a_0] \rightarrow 1_1$ is applied, and two neurons $\sigma_{t_1}$, $\sigma_{f_1}$ are generated. They have 8 synapses (1, $t_1$), (1, $f_1$), (4, $t_1$), (4, $f_1$), ($t_1$, out), ($f_1$, out), ($Cx_11, t_1$) and ($Cx_10, f_1$), where the first 6 synapses are produced by the heritage of the synapses (1, 0), (4, 0) and (0, out), respectively; the left two synapses are established by the synapse dictionary. The structure of the system after step $2n + 1$ is shown in Figure 6.

At step $2n + 2$, neurons $\sigma_{t_1}$ and $\sigma_{f_1}$ obtain one spike from neuron $\sigma_1$ respectively, then in the next step, only division rules can be applied in $\sigma_{t_1}$ and $\sigma_{f_1}$. So these two neurons are divided into four neurons with labels $t_2$ or $f_2$ corresponding to assignments $x_1 = 1$ and $x_2 = 1$, $x_1 = 1$ and $x_2 = 0$, $x_1 = 0$ and $x_2 = 1$, $x_1 = 0$ and $x_2 = 0$, respectively. The neuron $\sigma_{Cx_{11}}$ (encoding that $x_1$ appears in a clause) has synapses from it to neurons whose corresponding assignments have $x_1 = 1$. It means that assignments with $x_1 = 1$ satisfy clauses where $x_1$ appears. The structure of the system after step $2n + 3$ is shown in Figure 7.

The exponential workspace is produced by neuron division rules until $2^n$ neurons with labels $t_n$ or $f_n$ appear at step $4n - 1$. At step $4n - 2$, neuron $\sigma_3$ sends 2 spikes to neurons $\sigma_2$ and $\sigma_4$ (the rule $[a^5/a^2 \rightarrow a^2; 2n - 1]_3$ is applied at step $2n - 2$), while neuron $\sigma_1$ also send one spike to $\sigma_2$. So the spikes in neurons $\sigma_2$ and $\sigma_4$ are deleted by the rules $[a_3 \rightarrow \lambda]_2$ and $[a^2 \rightarrow \lambda]_4$. The auxiliary neurons $\sigma_1$
and $\sigma_2$ cannot supply spikes any more and the system pass to read the encoding of given instance. The structure of the system after step $4n - 1$ is shown in Figure 8.

**Input Stage:** The input module now consists of $2n + 2$ neurons, which are in the layers 1 – 3 as illustrated in Figure 5; $\sigma_{in}$ is the unique input neuron. The spikes of the encoding sequence $code(\gamma)$ are introduced into $\sigma_{in}$ one “package” by one “package”, starting from step 1. It takes $4n$ steps to introduce $(a^0)^{in}$ into neuron $\sigma_{in}$. At step $4n + 1$, the value of the first spike variable $\alpha_{11}$, which is the virtual symbol that represents the occurrence of the first variable in the first clause, enters into neuron $\sigma_{in}$. At the same time, neuron $\sigma_{d_0}$ sends 3 spikes to neuron $\sigma_{d_1}$ (the rule $[a^4/a^3 \rightarrow a^3; 4n]_{d_0}$ is used at the first step of the computation). At step $4n + 2$, the value of the spike variable $\alpha_{11}$ is replicated and sent to neurons $\sigma_{C_{x_i}}$, for all $i \in \{1, 2, \ldots, n\}$, while neuron $\sigma_{d_1}$ sends 3 auxiliary spikes to neurons $\sigma_{C_{x_1}}$ and $\sigma_{d_2}$. Hence, neuron $\sigma_{C_{x_1}}$ will contain 3, 4 or 5 spikes: if $x_1$ occurs in $C_1$, then neuron $\sigma_{C_{x_1}}$ collects 4 spikes; if $\neg x_1$ occurs in $C_1$, then it collects 5 spikes; if neither $x_1$ nor $\neg x_1$ occur in $C_1$, then it collects 3 spikes. Moreover, if neuron $\sigma_{C_{x_1}}$ has received 4 or 5 spikes, then it will be closed for $n - 1$ steps, according to the delay associated with the rules in it; on the other hand, if 3 spikes are received, then they are deleted and the neuron remains open. At step $4n + 3$, the value of the second spike variable $\alpha_{12}$ from neuron $\sigma_{in}$ is distributed to neurons $\sigma_{C_{x_i}}$, $2 \leq i \leq n$, where the spikes corresponding to $\alpha_{11}$ are deleted.
Figure 4.6: The structure of system $\Pi(\langle n, m \rangle)$ after step $2n + 1$

Figure 4.7: The structure of system $\Pi(\langle n, m \rangle)$ after step $2n + 3$
by the rules $[a \rightarrow \lambda]_{C_{x_i}}$ and $[a^2 \rightarrow \lambda]_{C_{x_i}}$, $2 \leq i \leq n$. At the same time, the 3 auxiliary spikes are duplicated and one copy of them enters into neurons $\sigma_{C_{x_2}}$ and $\sigma_{d_3}$, respectively. The neuron $\sigma_{C_{x_2}}$ will be closed for $n - 2$ steps only if it contains 4 or 5 spikes, which means that this neuron will not receive any spike during this period. In neurons $\sigma_{C_{x_i}}$, $3 \leq i \leq n$, the spikes represented by $\alpha_{12}$ are forgotten in the next step.

In this way, the values of the spike variables are introduced and delayed in the corresponding neurons until the value of the spike variable $\alpha_{1n}$ of the first clause and the 3 auxiliary spikes enter together into neuron $\sigma_{C_{x_n}}$ at step $5n + 1$ (note that neuron $\sigma_4$ also obtains 3 spikes from neuron $\sigma_{d_n}$ at the same step and will send one spike to the exponential workspace). At that moment, the representation of the first clause of $\gamma$ has been entirely introduced in the system, and the second clause starts to enter into the input module. In general, it takes $mn + 1$ steps to introduce the whole sequence $code(\gamma)$ in the system, and the input process is completed at step $4n + nm + 1$.

At step $4n + nm + 1$, the neuron $\sigma_{d_n}$ sends 3 spikes to neuron $\sigma_{d_1}$, while the auxiliary neuron $\sigma_{d_0}$ also sends a spike to the neuron $\sigma_{d_1}$ (the rule $[a \rightarrow a; nm - 1]_{d_0}$ is applied at step $4n + 1$). So neuron $\sigma_{d_1}$ contains 4 spikes, and in the next step these spikes are forgotten by the rule $[a^4 \rightarrow \lambda]_{d_1}$. It ensures that the system eventually halts.

Satisfiability Checking Stage: At step $5n + 1$, all the values of spike variables $\alpha_{1i}$
(1 \leq i \leq n), representing the first clause, have appeared in their corresponding neurons \sigma_{C,1} in the third layer, together with a copy of the 3 auxiliary spikes. In the next step, all the spikes contained in \sigma_{C,1} are duplicated and sent simultaneously to the pair of neurons \sigma_{C,1} and \sigma_{C,0} (1 \leq i \leq n) in the first layer of the satisfiability checking module. In this way, each neuron \sigma_{C,1} and \sigma_{C,0} receives 4 or 5 spikes when \( x_i \) or \( \neg x_i \) occurs in \( C_1 \), respectively, whereas it receives no spikes when neither \( x_i \) nor \( \neg x_i \) occurs in \( C_1 \).

In general, if neuron \( \sigma_{C,1} \) (1 \leq i \leq n) receives 4 spikes, then the literal \( x_i \) occurs in the current clause (say \( C_j \)), and thus the clause is satisfied by all those assignments in which \( x_i \) is true. Neuron \( \sigma_{C,0} \) will also receive 4 spikes, but they will be deleted during the next computation step. On the other hand, if neuron \( \sigma_{C,1} \) receives 5 spikes, then the literal \( \neg x_i \) occurs in \( C_j \), and the clause is satisfied by those assignments in which \( x_i \) is false. Since neuron \( \sigma_{C,1} \) is designed to process the case in which \( x_i \) occurs in \( C_j \), it will delete its 5 spikes. However, neuron \( \sigma_{C,0} \) will also have received 5 spikes, and this time it will send 3 spikes to those neurons which are bijectively associated with the assignments for which \( x_i \) is false (refer to the generation stage for the corresponding synapses). Note that, neuron \( \sigma_4 \) has 3 spikes at step \( 5n+1 \), the rule \([a^3 \rightarrow a; \lambda]_4\) is applied, one spike is duplicated and each enters into \( 2^n \) neurons with labels \( t_n \) or \( f_n \) at step \( 5n+3 \) because of the delay 1.

In this way, each neuron with label \( t_n \) or \( f_n \) receives 1 or 3k + 1 spikes (1 \leq k \leq n) at step \( 5n + 3 \). If one of neurons \( \sigma_t \) or \( \sigma_f \) (we assume the assignment of the neuron is \( t_1 t_2 \ldots t_{n-1} f_n \)) receives 1 spike, which means that none of the neurons \( \sigma_{C,1} \) and \( \sigma_{C,0} \) (1 \leq i \leq n) send spikes to this neuron, then the first clause \( C_1 \) is not satisfied by the assignment \( t_1 t_2 \ldots t_{n-1} f_n \) and this neuron can not be used to check whether other clauses \( C_j \) (2 \leq j \leq m) are satisfied or not. So the neuron with corresponding assignment \( t_1 t_2 \ldots t_{n-1} f_n \) is divided into two neurons \( \sigma_{t_{n+1}} \) and \( \sigma_{f_{n+1}} \) by the rule \([a]_4 \rightarrow [t]_{t_{n+1}} [f]_{f_{n+1}} \) is applied, and the two new neurons can not send any spike to the output neuron because they will have received the spikes by the rules \([a^{3k+1} \rightarrow \lambda]_{t_{n+1}}, [a^{3k+2} \rightarrow \lambda]_{f_{n+1}}, [a^{3k+1} \rightarrow \lambda]_{t_{n+1}}, [a^{3k+2} \rightarrow \lambda]_{f_{n+1}} \), with 0 \leq k \leq n. On the other hand, if a neuron (it is assumed that the assignment of the neuron is \( t_1 t_2 \ldots t_{n-1} t_n \)) receives 3k + 1 spikes from neurons \( \sigma_{C,1} \) and \( \sigma_{C,0} \), then these spikes will be forgotten, with the meaning that the clause \( C_1 \) is satisfied by the assignment \( t_1 t_2 \ldots t_{n-1} t_n \) (note that the number of spikes received in neurons with label \( t_n \) or \( f_n \) is not more than 3n + 1, because, without loss of generality, we assume that the same literal is not repeated and at most one of literals \( x_i \) or \( \neg x_i \), for any 1 \leq i \leq n, can occur in a clause; that is, a clause is a disjunction of at most n literals). This occurs in step \( 5n + 4 \). Thus, the satisfiability checking for the first clause has been done. The structure of the system after step \( 5n + 4 \) is shown in Figure 9 (it is assumed only the neuron with corresponding assignment \( t_1 t_2 \ldots t_{n-1} f_n \) is divided).

In a similar way, satisfiability checking for the next clause can proceed, and so on. Thus, the first \( m - 1 \) clauses can be checked to see whether there exist assignments that satisfy all of them. If there exist some assignments that satisfy
the first $m-1$ clauses, which means that their corresponding neurons never receive a spike during the satisfiability checking process of the $m-1$ clauses.

At step $4n + nm + 1$, the spike variable $\alpha_{m,n}$ of the last clause $C_m$ and the 3 auxiliary spikes (coming from neuron $\sigma_{d,a}$) enter into neuron $\sigma_{C,Cx_n}$. At the same moment, neuron $\sigma_4$ receives 3 spikes from neuron $\sigma_{d,a}$ and another 3 spikes from neuron $\sigma_3$ (the rule $[a^3 \rightarrow a^3]^2$ is applied at step $4n - 2$). So neuron $\sigma_4$ contains 6 spikes, and sends 2 spikes to the neurons with labels $t_n$, $f_n$, $t_{n+1}$ or $f_{n+1}$ at step $4n + nm + 3$ because of the delay 1. In this way, if neurons with labels $t_n$ and $f_n$ receive $3k + 2$ spikes $(1 \leq k \leq n)$, with the meaning that these neurons associated with the assignments satisfy all the clauses of $\gamma$, then the rules $[a^{3k+2}/a^2 \rightarrow a^2]_{t_n}$ and $[a^{3k+2}/a^2 \rightarrow a^2]_{f_n}$ can be applied, sending 2 spikes to neuron $\sigma_{out}$, respectively. On the other hand, if neurons with labels $t_n$ and $f_n$ receive only 2 spikes, or if neurons with labels $t_{n+1}$ and $f_{n+1}$ receive $3k + 2$ spikes $(0 \leq k \leq n)$, none of they can send spikes to the output neuron because they can not satisfy all the clauses of $\gamma$. In this way, the satisfiability checking module can complete its process at step $4n + nm + 4$.

Output Stage: From the above processes, it is not difficult to see that the neuron $\sigma_{out}$ receives spikes if and only if the formula $\gamma$ is true. At step $4n + nm + 5$, the output neuron sends exactly one spike to the environment if and only if the formula $\gamma$ is true.

According to the above four stages, one can see that the system correctly answers the question whether or not the formula $\gamma$ is satisfiable. The duration of
Conclusions and Remarks

the computation is polynomial in term of $n$ and $m$: the system sends one spike to the environment at step $4n + mn + 5$ if the answer is positive; otherwise, the system does not send any spike to the environment and halts in $4n + mn + 5$ steps.

The following is a comparison of the resources used in the systems constructed in this work and in [21]:

<table>
<thead>
<tr>
<th>Resources \ Systems</th>
<th>Systems from this work</th>
<th>Systems from [13]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial number of neurons</td>
<td>11</td>
<td>$4n + 7$</td>
</tr>
<tr>
<td>Initial number of spikes</td>
<td>20</td>
<td>9</td>
</tr>
<tr>
<td>Number of neuron labels</td>
<td>$10n + 7$</td>
<td>$6n + 8$</td>
</tr>
<tr>
<td>Size of synapse dictionary</td>
<td>$6n + 11$</td>
<td>$7n + 6$</td>
</tr>
<tr>
<td>Number of rules</td>
<td>$2n^2 + 26n + 26$</td>
<td>$n^2 + 14n + 12$</td>
</tr>
</tbody>
</table>

From the above comparison, it is easy to see that the amount of necessary resources for defining each system in this work is polynomial with respect to $n$. Note that the sets of rules associated with the system $\Pi(\langle n, m \rangle)$ are recursive. Hence, the family $\Pi = \{\Pi(\langle n, m \rangle) \mid n, m \in \mathbb{N}\}$ is polynomially uniform by deterministic Turing machines.

The result of [21] is improved in the sense that a constant number of initial neurons (instead of linear number) are used to construct the systems for efficiently solving $\text{SAT}$ problem, and the neuron budding rules are not used.

### 4.4 Conclusions and Remarks

In this work, a uniform family of SN $P$ systems with only neuron division (not using neuron budding) is constructed for efficiently solving $\text{SAT}$ problem, which answers an open problem posed in [21]. It is interesting that a constant number of initial neurons are used to construct the systems.

There remain many open problems and research topics about neuron division and neuron budding. As we know, all $\text{NP}$ problems can be reduced to an $\text{NP}$-complete problem in a polynomial time. In principle, if a family of $P$ systems can efficiently solve an $\text{NP}$-complete problem, then this family of $P$ systems can also efficiently solve all $\text{NP}$ problems. But, until now, it remains open how a family of $P$ systems can be designed to efficiently compute the reduction from an $\text{NP}$ problem to an $\text{NP}$-complete problem. Before this open problem is solved, it is still interesting to give efficient solutions to computationally hard problems in the framework of SN $P$ systems with neuron division.

It is worth investigating the computation power of SN $P$ systems with only neuron budding rules without neuron division rules. Neuron budding can result in a polynomial number of neurons in a polynomial time, and each neuron has a polynomial number of spikes. It is hard to believe that SN $P$ systems with only neuron budding rules can efficiently solve computationally hard problems unless we happen to find a proof for $P = \text{NP}$.
Acknowledgements

The work of J. Wang and L. Pan was supported by National Natural Science Foundation of China (Grant Nos. 61033003, 30870826, and 60703047), China Scholarship Council, HUST-SRF (2007Z015A), and Natural Science Foundation of Hubei Province (2008CDB113 and 2008CDB180).