Small universal simple spiking neural P systems with weights

ZENG XiangXiang[1,2], PAN LinQiang[1]* & PÉREZ-JIMÉNEZ Mario J.[2]

[1]*Key Laboratory of Image Processing and Intelligent Control, School of Automation, Huazhong University of Science and Technology, Wuhan* 430074*, China;*

[2]*Department of Computer Science and Artificial Intelligence, University of Sevilla, Avda. Reina Mercedes s/n, Sevilla* 41012*, Spain*

Abstract Spiking neural P systems with weights (WSN P systems, for short) are a new variant of spiking neural P systems, where the rules of a neuron are enabled when the potential of that neuron equals a given value. It is known that WSN P systems are universal by simulating register machines. However, in these universal systems, no bound is considered on the number of neurons and rules. In this work, a restricted variant of WSN P systems is considered, called simple WSN P systems, where each neuron has only one rule. The complexity parameter, the number of neurons, to construct a universal simple WSN P system is investigated. It is proved that there is a universal simple WSN P system with 48 neurons for computing functions; as generator of sets of numbers, there is an almost simple (that is, each neuron has only one rule except that one neuron has two rules) and universal WSN P system with 45 neurons.

Keywords bio-inspired computing, membrane computing, P system, spiking neural P system, universal computing device

1 Introduction

Membrane computing is one of the recent branches of natural computing [1,2], which was initiated by Păun [3] and has developed very rapidly (already in 2003, ISI considered membrane computing as "fast emerging research area in computer science", see http://esi-topics.com). The aim of membrane computing is to abstract computing ideas (data structures, operations with data, computing models, etc.) from the structure and the functioning of a single cell or complexes of cells such as tissues and organs. The obtained models are distributed and parallel computing devices, called P systems.

Spiking neural P systems (SN P systems, for short) were introduced as a new class of P systems [4], with the aim of incorporating specific ideas from spiking neurons into membrane computing. In short, an SN P system consists of a set of neurons placed in the nodes of a directed graph, where neurons send signals (spikes, denoted by the symbol a in what follows) along synapses (arcs of the graph). The neurons

*Corresponding author (email: lqpan@mail.hust.edu.cn)

Table 1 Small universal SN P systems and WSN P systems, where each rule can only produce one spike (or one unit potential) at each time step

Systems	Number of (types of) neurons	Number of (types of) rules	Applicability of rules
SN P system for computing functions [9]	(8) 84	(9) 113	Regular sets
SN P system as number generators [9]	(8) 76	(9) 109	Regular sets
SN P system for computing functions [10]	(8) 67	(9) 96	Regular sets
SN P system as number generators [10]	(8) 63	(9) 93	Regular sets
SN P system for computing functions [11]	(7) 11	(unbounded) unbounded	Regular sets
WSN P system for computing functions (Section 4)	(2) 48	(2) 48	Thresholds
WSN P system as number generators (Section 5)	(3) 45	(3) 46	Thresholds

contain spiking and forgetting rules for emitting spikes and forgetting spikes. Spiking rules are of the form $E/a^c \to a; d$, where E is a regular expression over $\{a\}$ and c, d are natural numbers, $c \geqslant 1,\ d \geqslant 0$. If a neuron contains k spikes such that $a^k \in L(E),\ k \geqslant c$, then, by applying a firing rule, it can consume c spikes and produce one spike, after a delay of d steps. This spike is sent to all neurons connected by an outgoing synapse from the neuron where the rule was applied. Forgetting rules are of the form $a^s \to \lambda$, with the meaning that $s \geqslant 1$ spikes are removed if the neuron contains exactly s spikes. The system works in a synchronized manner: in each time unit, (1) the system works in parallel as a whole, but sequentially at the level of each single neuron: at most one rule is applied in each neuron at every computation step; (2) if at a given time two or more rules can be applied in a neuron, then only one of them is chosen in a nondeterministic way. One of the neurons is considered to be the output neuron, and its spikes are also sent to the environment. The result of a computation is defined as the time distance between the first two spikes emitted by the output neuron.

A neuron with only one rule is said to be simple. A simple SN P system is an SN P system such that all neurons are simple. An almost simple SN P system is an SN P system such that all neurons are simple except for one neuron.

An SN P system such that each neuron has the same set of rules is said to be homogeneous. A semi-homogeneous SN P system is an SN P system such that all neurons have the same set of rules except for one neuron.

In SN P systems, the applicability of each rule is determined by checking the number of spikes in the neuron against a regular set associated with the rule. It is proved that it is at least NP-hard to decide whether a rule can be applied [5]. In order to decide the applicability of rules in an easy way, spiking neural P systems with weights (WSN P systems, for short) were introduced as a variant of SN P systems [6]. Instead of counting spikes as in a usual SN P system, each neuron in a WSN P system contains a potential, which can be expressed by a computable real number. Each neuron fires when its potential equals a given value (called threshold). The execution of a rule consumes a part of the potential and produces a unit potential. This unit potential passes to neighboring neurons multiplied by the weights of synapses.

It is a natural and well investigated topic in computer science to look for small universal computing devices of various types. This topic was also considered in membrane computing, e.g. [7,8]. Particularly, some results on small universal computing devices in the framework of SN P systems are listed in Table 1. In [9], a universal SN P system with 84 neurons is constructed as a device of computing functions, where the number of rules is 113, 8 types of neurons and 9 types of rules are used; as generators of sets of numbers, a universal SN P system with 76 neurons is given, where the number of rules is 109, 8 types of neurons and 9 types of rules are used. This result is improved both in the number of neurons and the number of rules in [10]. If arbitrarily many rules and types of rules are used, the number of neurons can be reduced to 11 as a device for computing functions [11].

In this work, the problem of constructing universal WSN P systems with a small number of neurons

is investigated. Specifically, a universal simple and almost homogeneous WSN P system with 48 neurons is constructed for computing functions; as generator of sets of numbers, a universal and almost simple WSN P system with 45 neurons is constructed. In the systems constructed in this work, the neurons are quite "simple" in the sense that each neuron has only one rule. Furthermore, each system constructed in this work is almost homogeneous in the sense that all neurons have the same set of rules except for one neuron. The results given in this work are of interest with the following interpretation: although the neurons are simple and homogeneous, a network of neurons can be powerful—"complete (Turing) creativity" by cooperating with each other.

2 Prerequisites

It is useful for the reader to have some familiarity with (basic elements of) language theory, e.g., from [12], as well as basic membrane computing, e.g., [13]. We introduce here only a few notations and the definitions related to universal register machines.

A register machine is a tuple $M = (m, H, l_0, l_h, I)$, where m is the number of registers, H is the set of instruction labels, l_0 is the start label, l_h is the halt label (assigned to instruction HALT), and I is the set of instructions; each label from H labels exactly one instruction from I, thus precisely identifying it. The instructions are of the following forms:

1) $l_i : (ADD(r), l_j, l_k)$ (add 1 to register r and then go to one of the instructions with labels l_j, l_k non-deterministically chosen),

2) $l_i : (SUB(r), l_j, l_k)$ (if register r is non-empty, then subtract 1 from it and go to the instruction with label l_j, otherwise go to the instruction with label l_k),

3) $l_h : HALT$ (the halt instruction).

A register machine M generates a set $N(M)$ of numbers in the following way: the machine starts with all registers being empty (i.e., storing the number zero); the machine applies the instruction with label l_0 and continues to apply instructions as indicated by the labels (and made possible by the contents of registers); if it reaches the halt instruction, then the number n present in specified register r_0 at that time is said to be generated by M. If the computation does not halt, then no number is generated. It is known that register machines generate all sets of numbers which are Turing computable (see, e.g., [14]).

A register machine can also compute any Turing computable function: the arguments are introduced in specified registers $r_1, \ldots, r_k$ (without loss of generality, it can be assumed that the first k registers are used), the register machine starts with the instruction with label l_0; if it stops (with the instruction with label l_h), then the value of the function is placed in another specified register r_t, with all registers different from r_t being empty. The partial function computed in this way is denoted by $M(n_1, n_2, \ldots, n_k)$. In the computing mode, register machines can be considered deterministic, without losing the Turing completeness; in this case, the ADD instructions $l_i : (ADD(r), l_j, l_k)$ have $l_j = l_k$ (hence, the instruction can be written in the form $l_i : (ADD(r), l_j)$).

In [15], register machines are used for computing functions, with the universality defined as follows: let $(\varphi_0, \varphi_1, \ldots)$ be a fixed admissible enumeration of the unary partial recursive functions. A register machine M_u is said to be universal if there is a recursive function g such that for all natural numbers x, y we have $\varphi_x(y) = M_u(g(x), y)$. In [15], several universal register machines have been constructed for computing functions, with the input introduced in registers 1 and 2, and the result obtained in register 0. A specific universal register machine M_u from [15] is given in Figure 1. The construction of small universal WSN P systems in Sections 4 and 5 is based on the specific universal register machine M_u given in Figure 1.

We use the following convention: when comparing the power of two number generating/accepting devices D_1 and D_2, number zero is ignored; that is, we write $N(D_1) = N(D_2)$ if and only if $N(D_1) - \{0\} = N(D_2) - \{0\}$ (this corresponds to the usual practice of ignoring the empty string in language and automata theory).

l_0 : (SUB(1), l_1, l_2), l_1 : (ADD(7), l_0),

l_2 : (ADD(6), l_3), l_3 : (SUB(5), l_2, l_4),

l_4 : (SUB(6), l_5, l_3), l_5 : (ADD(5), l_6),

l_6 : (SUB(7), l_7, l_8), l_7 : (ADD(1), l_4),

l_8 : (SUB(6), l_9, l_0), l_9 : (ADD(6), l_{10}),

l_{10} : (SUB(4), l_0, l_{11}), l_{11} : (SUB(5), l_{12}, l_{13}),

l_{12} : (SUB(5), l_{14}, l_{15}), l_{13} : (SUB(2), l_{18}, l_{19}),

l_{14} : (SUB(5), l_{16}, l_{17}), l_{15} : (SUB(3), l_{18}, l_{20}),

l_{16} : (ADD(4), l_{11}), l_{17} : (ADD(2), l_{21}),

l_{18} : (SUB(4), l_0, l_h), l_{19} : (SUB(0), l_0, l_{18}),

l_{20} : (ADD(0), l_0), l_{21} : (ADD(3), l_{18}),

l_h : $HALT$

Figure 1 A universal register machine.

3 WSN P systems

WSN P systems were introduced in [6]. Here, the definition of WSN P systems is recalled.

A WSN P system, of degree $m \geqslant 1$, is a construct of the form

$$\Pi = (\sigma_1, \ldots, \sigma_m, syn, in, out),$$

where:

1) $\sigma_1, \ldots, \sigma_m$ are neurons, of the form $\sigma_i = (p_i, R_i)$, $1 \leqslant i \leqslant m$, where a) $p_i \in \mathbb{R}_c$ ($\mathbb{R}_c$ is the set of computable real numbers) is the initial potential in σ_i; b) R_i is a finite set of spiking rules of the form $T_i/d_s \to 1$, $s = 1, 2, \ldots, n_i$ for some $n_i \geqslant 1$, where $T_i \in \mathbb{R}_c$, $T_i \geqslant 1$, is the firing threshold potential of neuron σ_i, and $d_s \in \mathbb{R}_c$ with the restriction $0 < d_s \leqslant T_i$;

2) $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\} \times \mathbb{R}_c$ are synapses between neurons, where $i \neq j$, $w \neq 0$ for each $(i, j, w) \in syn$, and for each $(i, j) \in \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ there is at most one synapse (i, j, w) in syn;

3) $in, out \in \{1, 2, \ldots, m\}$ indicate the input and output neurons, respectively.

The spiking rules are applied as follows. Assume that at a given moment, neuron σ_i has a potential p. If $p = T_i$, then any rule $T_i/d_s \to 1 \in R_i$ can be applied. The execution of this rule consumes an amount of d_s of the potential (thus leaving the potential $T_i - d_s$) and prepares one unit potential (also called a spike) to be delivered to all the neurons σ_j such that $(i, j, w) \in syn$. Specifically, each of these neurons σ_j receives a quantity of potential equal to w, which is added to the existing potential in σ_j. Note that w can be positive or negative, hence the potential of the receiving neuron is increased or decreased depending on w. The potential emitted by a neuron σ_i passes immediately to all neurons σ_j such that $(i, j, w) \in syn$; that is, the transition of potential takes no time. If a neuron σ_i spikes and it has no outgoing synapse, then the potential emitted by neuron σ_i is lost.

Note that each neuron σ_i has only one fixed threshold potential T_i. If a neuron has the potential equal to its firing threshold potential, then all rules associated with this neuron are enabled, and only one of them is non-deterministically chosen to be applied. In each step (a global clock is assumed, marking the time for the whole system, hence the functioning of the system is synchronized), each neuron uses at most one rule, non-deterministically chosen among its rules, provided that its potential equals the firing threshold, but all neurons that have applicable rules must choose and apply a rule.

If neuron σ_i has a potential p such that $p < T_i$, then the neuron σ_i returns to the resting potential 0. If neuron σ_i has a potential p such that $p > T_i$, then potential p remains unchanged.

To sum up, if neuron σ_i has potential p and receives potential k at step t, then at step $t+1$ it has potential p', where:

$$p' = \begin{cases} k, & \text{if } p < T_i, \\ p - d_s + k, & \text{if } p = T_i \text{ and rule } T_i/d_s \to 1 \text{ is applied}, \\ p + k, & \text{if } p > T_i. \end{cases}$$

The configuration of the system is described by the distribution of potentials in neurons. Thus, the initial configuration of the system is the tuple $\langle p_1, p_2, \ldots, p_m \rangle$. Using the rules as described above, one can define transitions among configurations. Any sequence of transitions starting from the initial configuration is called a computation. A computation halts if it reaches a configuration where no rule can be applied.

In order to compute a function $f : \mathbb{N}^k \to \mathbb{N}$ (where $\mathbb{N}$ is the set of natural numbers), k natural numbers $n_1, \ldots, n_k$ are introduced into the system by "reading" from the environment a binary sequence $z = 10^{n_1-1}10^{n_2-1}1 \cdots 10^{n_k-1}1$. This means that the input neuron of the system receives a spike at each step corresponding to a digit 1 from string z and no spike otherwise. Note that $k+1$ spikes are exactly inputted; that is, it is assumed that no further spike is coming to the input neuron after the last spike. The result of the computation is encoded in the time distance between the first two spikes emitted by the system with the restriction that the system outputs exactly two spikes and halts (immediately after the second spike), hence it produces a spike train of the form $0^b10^{r-1}1$, for some $b \geqslant 0$ and with $r = f(n_1, \ldots, n_k)$ (the system outputs no spike for a non-specified number of steps from the beginning of the computation until the first spike).

SN P systems can also be used as number generators as in [4]. An SN P system starts working in its initial configuration; because of the non-determinism in using the spiking rules, several computations are possible; any halting computation provides a result, in the form of the number of time units between the first two steps when any spike exits the system. In this case, an SN P system Π_u is universal if, given a fixed admissible enumeration of the unary partial recursive functions, $(\varphi_0, \varphi_1, \ldots)$, there is a recursive function g such that for each natural number x, if we input the number $g(x)$ in Π_u, by "reading" the sequence $10^{g(x)-1}1$ from the environment, the set of numbers generated by the system is equal to $\{n \in N \mid \varphi_x(n) \text{ is defined}\}$. Specifically, the strategy followed by the universal system in the case of generating numbers is as follows: (1) read the string $10^{g(x)-1}1$ from the environment and load $2g(x)$ spikes in neuron σ_1; (2) load neuron σ_2 non-deterministically with $f(n)$ spikes (corresponding to that number n is input in register 2), where n is an arbitrary natural number; at the same time, output the spike train $0^b10^{n-1}1$ (hence the number n), $b \geqslant 0$; (3) if system Π_u with $2g(x)$ spikes in neuron σ_1 and $f(n)$ spikes in neuron σ_2 halts, then n is introduced in the set of generated numbers.

In the next sections, WSN P systems are represented graphically, which may be easier to understand than a symbolic representation. An oval with the initial potential and spiking rules inside is used to represent a neuron, and arrows between these ovals represent the synapses; numbers will mark these arrows, indicating the weights. The input neuron has an incoming arrow and the output neuron has an outgoing arrow, suggesting their communication with the environment. When the weight on a synapse is one, it is omitted in the graphical representation.

4 A small universal WSN P system for computing functions

In this section, a small universal WSN P system for computing functions is constructed.

Theorem 1. There exists a universal simple WSN P system with 87 neurons for computing functions.

Proof. We design a specific WSN P system Π simulating the universal register machine M_u shown in Figure 1. The simulation is done as follows. Neurons are associated with each register and with each label of an instruction of the machine. If a register r of M_u contains a number n, then the associated

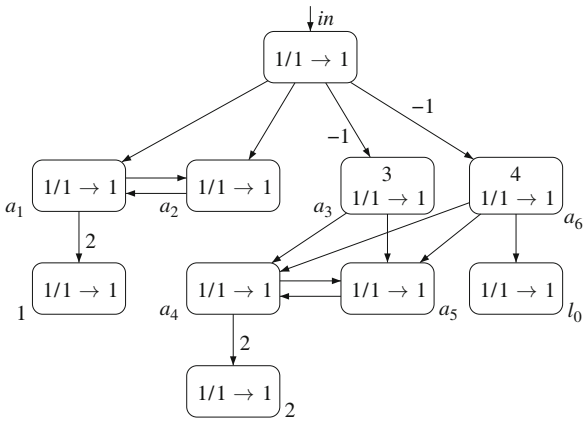

Figure 2 Module INPUT.

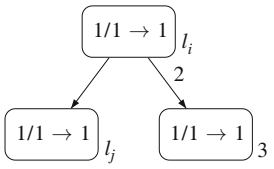

Figure 3 Module ADD (simulating $l_i : (ADD(r), l_j)$).

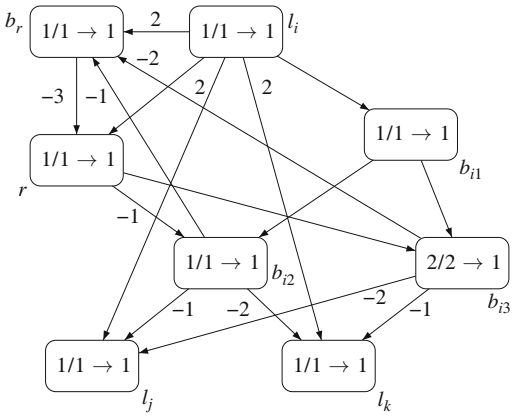

Figure 4 Module SUB (simulating $l_i : (SUB(r), l_j, l_k)$).

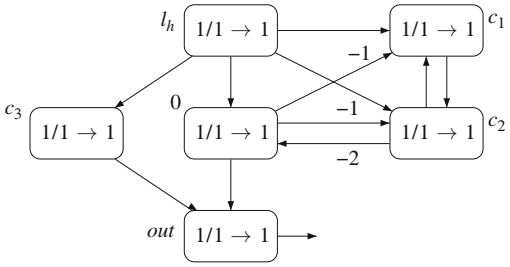

Figure 5 Module OUTPUT.

neuron σ_r will contain the potential $2n$. Specifically, the system Π is composed of the following four types of modules.

(i) An INPUT module is shown in Figure 2. It is used for loading neurons σ_1 and σ_2 with potentials $2g(x)$ and $2y$ which represent the numbers $g(x)$ and y, respectively, and for activating neuron σ_{l_0} associated with the starting instruction of the register machine M_u by introducing one unit potential into neuron σ_{l_0}.

(ii) An ADD module for simulating a deterministic ADD instruction: $l_i : (ADD(r), l_j)$ is presented in Figure 3.

(iii) A SUB module for simulating a SUB instruction $l_i : (SUB(r), l_j, l_k)$ is shown in Figure 4. In addition to three neurons σ_{l_i}, σ_{l_j} and σ_{l_k} for three labels l_i, l_j, l_k, and two neurons σ_r and σ_{b_r} for register r, each such module contains three auxiliary neurons $\sigma_{b_{i1}}$, $\sigma_{b_{i2}}$ and $\sigma_{b_{i3}}$.

(vi) An OUTPUT module for outputting the result placed in register 0 is shown in Figure 5.

The input of the system is encoded with the spike train $10^{g(x)-1}10^{y-1}1$. In the initial configuration of Π all neurons are empty. Neuron σ_{in} of the INPUT module reads the input spike train. The first unit potential entering neuron σ_{in} passes to neurons σ_{a_1} and σ_{a_2}, which will feed each other. Neuron σ_1 receives from neuron σ_{a_1} as many potential units as the double number of steps between the first two input potentials (the potentials from neuron σ_{a_1} to neuron σ_1 are amplified by the synapse weight 2), and after that neuron σ_{a_1} gets "over flooded" by the second input potential unit and is blocked. Neuron σ_{a_3} spikes after its potential is decreased by the first two potentials -1 from neuron σ_{in}; neurons σ_{a_4} and σ_{a_5} receive one unit potential from neuron σ_{a_3}, respectively. In the next step, neuron σ_{a_4} starts to send potentials to neuron σ_2 until receiving a potential from neuron σ_{a_6} (the potential sent out by neuron σ_{a_4} is amplified by the synapse weight 2). In this way, potentials $2g(x)$ and $2y$ are loaded in neurons σ_1 and

σ_2, respectively. When neuron σ_{l_0} (associated with the starting instruction l_0 of the register machine) gets potential 1 from neuron σ_{a_6} (neuron σ_{a_6} spikes after it receives the third potential -1; that is, after the "reading" of the input spike train finishes), the simulation work of the system is triggered. In general, the simulation of an ADD or SUB instruction starts by introducing potential 1 in the neuron associated with the corresponding instruction. As we will see below, when an instruction is simulated by an ADD or SUB module, the system cannot activate the simulation of another instruction at the same time.

Assume that system Π is at a step when it has to simulate an ADD instruction $l_i : (ADD(r), l_j)$. At this moment, neuron σ_{l_i} has potential 1 and other neurons have potential 0, except for neurons associated with registers. Having potential 1 inside, neuron σ_{l_i} spikes. Neuron σ_r receives potential 2 and the potential in neuron σ_r increases by 2, which means that the number stored in register r increases by one. Neuron σ_{l_j} receives potential 1, and in the next step it fires, which means that the system Π starts to simulate the next instruction l_j. So the ADD instruction $l_i : (ADD(r), l_j)$ is correctly simulated by the system Π.

The initial instruction with label l_0 is a SUB instruction. Assume that system Π is in a step t when it has to simulate a SUB instruction $l_i : (SUB(r), l_j, l_k)$. At this moment, neuron σ_{l_i} has potential 1 and the other neurons have potential 0 (as in the SUB module associated with the initial instruction l_0), except for those neurons associated with the registers. At step $t+1$, neuron σ_{l_i} fires; neuron σ_r receives potential 1; each neuron σ_{b_r}, σ_{l_j}, σ_{l_k} receives potential 2; and neuron $\sigma_{b_{i1}}$ receives potential 1. At step $t+2$, neuron $\sigma_{b_{i1}}$ fires, neurons $\sigma_{b_{i2}}$ and $\sigma_{b_{i3}}$ receive potential 1 from neuron $\sigma_{b_{i1}}$. For neuron σ_r, there are the following two cases:

(i) The potential of neuron σ_r at step t is 0 (that is, the number stored in neuron σ_r is 0). At step $t+1$, neuron σ_{l_i} fires. At step $t+2$, neuron σ_r has potential 1 (it has received potential 1 from neuron σ_{l_i} at the previous step), and it spikes by rule $1/1 \to 1$. At step $t+2$, neuron $\sigma_{b_{i2}}$ receives potential 1 from neuron $\sigma_{b_{i1}}$ and potential -1 from neuron σ_r, so it has potential 0; neuron $\sigma_{b_{i3}}$ receives potential 2 (one unit of potential from neuron $\sigma_{b_{i1}}$, another one from neuron σ_r), and it spikes at step $t+3$. Receiving potential -1 from neuron $\sigma_{b_{i3}}$, neuron σ_{l_k} has potential 1 (it received potential 2 from neuron σ_{l_i} at step $t+1$) and becomes active, starting to simulate the instruction l_k of M_u. At step $t+3$, neurons σ_{l_j} and σ_{b_r} receive potential -2 from neuron $\sigma_{b_{i3}}$, both of them have potential 0 and cannot spike.

(ii) The potential of neuron σ_r is $2n$ ($n > 0$) at step t (that is, the number stored in neuron σ_r is n). At step $t+1$, neuron σ_{l_i} fires. At step $t+2$, neuron σ_r has potential $2n+1$, which is greater than its threshold, and will keep unchanged. At step $t+2$, neuron $\sigma_{b_{i3}}$ receives potential 1 from neuron $\sigma_{b_{i1}}$, which is less than its threshold, hence it will not spike and its potential will vanish to 0 at step $t+3$. At step $t+2$, neuron $\sigma_{b_{i2}}$ receives potential 1 from neuron $\sigma_{b_{i1}}$ and it spikes at step $t+3$. Receiving potential -1 from neuron $\sigma_{b_{i2}}$, neuron σ_{l_j} has potential 1 (it received potential 2 from neuron σ_{l_i} at step $t+1$) and becomes active, starting to simulate the instruction l_j of M_u. Note that at step $t+3$, neuron σ_{b_r} receives potential -1 from neuron $\sigma_{b_{i2}}$, it has potential 1 and spikes; neuron σ_r receives potential -3 from neuron σ_{b_r}; in this way it correctly ends with potential $2n-2$, which simulates that the number stored in register r is decreased by one.

The system Π starts from σ_{l_i} and ends in σ_{l_j}, if the register r is non-empty and the number stored in register r is decreased by one. If the register r is empty, then the system Π starts from σ_{l_i} and ends in σ_{l_k}. So the SUB instruction $l_i : (SUB(r), l_j, l_k)$ is correctly simulated by the system Π.

The computation result of the register machine is stored in register 0. The system Π outputs the result by means of the OUTPUT module shown in Figure 5. Assume that the computation in M_u halts, which means that the halt instruction l_h is reached. This means that neuron σ_{l_h} receives potential 1 and fires by rule $1/1 \to 1$. At that moment, neuron σ_0 has potential $2n$, for the number $n \geqslant 0$ stored in register 0 of M_u. When σ_{l_h} fires, each neuron σ_{c_1}, σ_{c_2}, σ_{c_3} receives potential 1; neuron σ_0 receives potential 1, changing its potential to $2n+1$. Suppose that this is step t. At step $t+1$, neuron σ_{c_3} spikes; neuron σ_{out} receives potential 1 from neuron σ_{c_3}, and spikes at step $t+2$ (this is the first spike sent out by system Π).

From step $t+1$ on, neurons σ_{c_1} and σ_{c_2} send potential 1 to each other, consuming one unit potential. This process continues until they receive potential -1 from neuron σ_0. During this process, at each step, neuron σ_0 receives potential -2 from neuron σ_{c_2}, which corresponds to decreasing by one the number

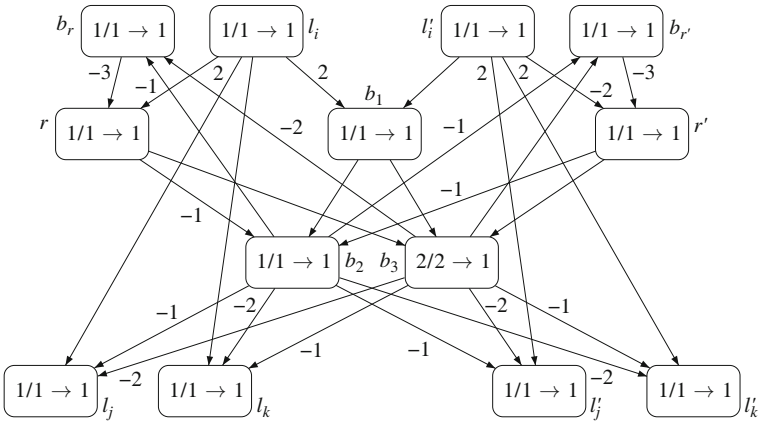

Figure 6 A part of construction associated with two SUB instructions acting on different registers.

stored in register 0. At step $t + n + 1$, neuron σ_0 has potential 1 and spikes; neurons σ_{c_1} and σ_{c_2} have potential 0 after receiving potential -1 from neuron σ_0. Receiving potential 1 from σ_0 at step $t + n + 1$, the output neuron σ_{out} spikes again at step $t + n + 2$, and the system sends the second spike to the environment. The interval between these two spikes sent out by the system is $(t + n + 2) - (t + 2) = n$, which is exactly the number stored in register 0 of M_u at the moment when the computation of M_u halts.

If the number stored in register 0 is 0 when register machine M_u halts, then at step $t+1$, neuron σ_{out} has potential 2, which is greater than its threshold potential 1. In this case, neuron σ_{out} is blocked, and system Π sends no spike to the environment. Recall that 0 is ignored when we investigate the power of computing devices.

From the above description of the modules and their work, it is clear that the register machine M_u is correctly simulated by the system Π.

We can check that each neuron in the system Π has only one rule; that is, the system Π is simple. The system Π has

1) 16 neurons for the 8 registers: $\sigma_0, \ldots, \sigma_7, \sigma_{b_0}, \ldots, \sigma_{b_7}$,
2) 23 neurons for the 23 labels: $\sigma_{l_h}, \sigma_{l_i}$ $(0 \leqslant i \leqslant 21)$,
3) 39 neurons for the 13 SUB instructions: $\sigma_{b_{i1}}, \sigma_{b_{i2}}, \sigma_{b_{i3}}$,
4) 5 neurons in the INPUT module: $\sigma_{in}, \sigma_{a_1}, \ldots, \sigma_{a_4}$,
5) 4 neurons in the OUTPUT module: $\sigma_{out}, \sigma_{c_1}, \sigma_{c_2}, \sigma_{c_3}$.

So the system Π has 87 neurons in total.

In what follows, we show that the number of neurons in system Π can be decreased by exploiting some particularities of the register machine M_u. Particularly, in the new system, there is only one neuron with the spiking rule $2/2 \to 1$.

Theorem 2. There exists a universal simple and semi-homogeneous WSN P system with 48 neurons for computing functions.

Proof. We show that all SUB modules can share a neuron with the spiking rule $2/2 \to 1$; that is, each neuron has the spiking rule $1/1 \to 1$ except for a neuron with the spiking rule $2/2 \to 1$. Therefore, the system is semi-homogeneous. Specifically, all SUB modules can share three auxiliary neurons σ_{b_1}, σ_{b_2} and σ_{b_3}: (1) remove neurons $\sigma_{b_{i1}}$, $\sigma_{b_{i2}}$ and $\sigma_{b_{i3}}$ from the SUB module associated with the instruction $l_i : (SUB(r), l_j, l_k)$; (2) add neurons σ_{b_1}, σ_{b_2} and σ_{b_3}; (3) for each neuron σ_t (except for neurons $\sigma_{b_{i1}}$, $\sigma_{b_{i2}}$ and $\sigma_{b_{i3}}$) in the SUB module associated with the instruction $l_i : (SUB(r), l_j, l_k)$, if there is a synapse (t, b_{ij}) (resp. (b_{ij}, t)), then a synapse (t, b_j) (resp. (b_j, t)) is added, where $j = 1, 2, 3$. We show that there is no undesired effect appears by sharing the three auxiliary neurons when a SUB instruction $l_i : (SUB(r), l_j, l_k)$ is simulated. To this aim, we consider the following two cases.

i) For any other SUB instruction $l_i' : (SUB(r'), l_j', l_k')$ with $r' \neq r$, the construction associated with these two SUB instructions l_i and l_i' is shown in Figure 6. Let us examine the work of the construction

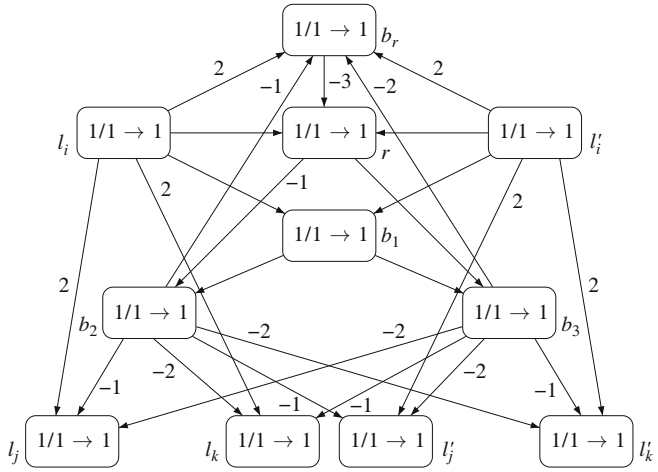

Figure 7 A part of construction associated with two SUB instructions acting on the same register.

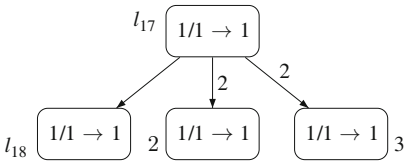

Figure 8 A module for consecutive ADD instructions.

from Figure 6. Assume that the system starts to simulate the SUB instruction $l_i : (SUB(r), l_j, l_k)$ at time t, then $\sigma_{l'_j}$, $\sigma_{l'_k}$, and $\sigma_{b_{r'}}$ will receive potential -1 or potential -2 from neuron σ_{b_2} (or from neuron σ_{b_3}, if register r is empty) at time $t+3$. However, after receiving these negative potentials, neurons $\sigma_{l'_j}$, $\sigma_{l'_k}$ and $\sigma_{b_{r'}}$ have less potentials than their corresponding firing thresholds, so their potentials return to 0 at the next step. Hence, after the simulation of $l_i : (SUB(r), l_j, l_k)$, all the neurons will go back to their initial state except for neuron σ_r. The next simulation can continue correctly.

ii) For any other SUB instruction $l'_i : (SUB(r), l'_j, l'_k)$ with $r' = r$, the construction associated with these two SUB instructions l_i and l'_i is shown in Figure 7. Similar to the case (i), we can check that there is no undesired effect appears by sharing the three auxiliary neurons.

In what follows, we present the modules that allow further decreasing the number of neurons but we do not go into details explaining their work.

Let us observe that the sequence of two consecutive ADD instructions $l_{17} : (ADD(2), l_{21})$ and $l_{21} : (ADD(3), l_{18})$ has no other instruction addressing label l_{21}, which can be simulated by the module from Figure 8. In this way, a neuron associated with l_{21} is saved.

The module from Figure 9 can simulate the consecutive ADD-SUB instructions $l_5 : (ADD(5), l_6)$ and $l_6 : (SUB(7), l_7, l_8)$. A similar module can be constructed to simulate the consecutive ADD-SUB instructions $l_9 : (ADD(6), l_{10})$ and $l_{10} : (SUB(4), l_0, l_{11})$. So two neurons (associated with the labels l_6 and l_{10}) are saved.

From the above description about the decrease of the number of neurons, we can check that 39 neurons are saved. In the modified system, each neuron has the same spiking rule $1/1 \to 1$ except for one neuron with the spiking rule $2/2 \to 1$.

5 A small universal WSN P system used as a number generator

In this section, a small universal WSN P system for generating numbers is constructed.

Theorem 3. There exists a universal almost simple WSN P system with 45 neurons that can be used as a number generator.

Proof. We prove this theorem by modifying the proof of Theorem 2. Following the definition of generating numbers given Section 2, the WSN P system Π_u constructed here works as follows. (1) read the string $10^{g(x)-1}1$ from the environment and load $2g(x)$ spikes in neuron σ_1; (2) load neuron σ_2 nondeterministically with $2n$ spikes (corresponding to that number n is input in register 2), where n is an arbitrary natural number; at the same time, output the spike train $0^b10^{n-1}1$ (hence the number n),

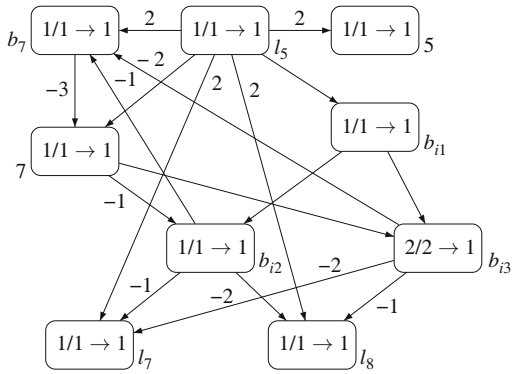

Figure 9 A module for consecutive ADD-SUB instructions.

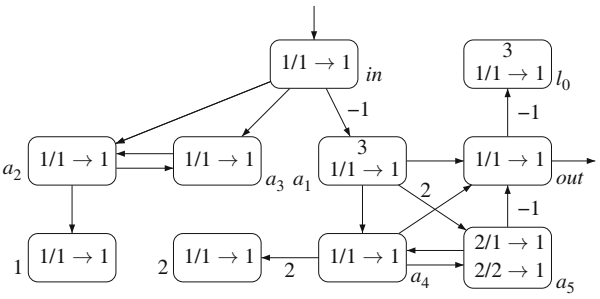

Figure 10 Module INPUT-OUTPUT.

where $b \geqslant 0$; (3) if system Π_u with $2g(x)$ spikes in neuron σ_1 and $2n$ spikes in neuron σ_2 halts, then n is introduced in the set of generated numbers. To this aim, the INPUT module and OUTPUT module given in Section 4 are combined as the INPUT-OUTPUT module, which is shown in Figure 10.

The INPUT-OUTPUT module starts "reading" the spike train $10^{g(x)-1}1$ from the environment and loads potential $2g(x)$ in neuron σ_1. After "reading" the second spike from the environment, the module passes to load neuron σ_2 non-deterministically with an arbitrary potential $2n$ (corresponding to natural number n); at the same time, neuron σ_{out} outputs number n in the form of a spike train $0^b10^{n-1}1$, for some $b \geqslant 0$. After neuron σ_{out} sends the second spike to the environment and to neuron σ_{l_0}, neuron σ_{l_0} is activated to start the simulation of the register machine M_u from Figure 1, with $g(x)$ in register 1 and n in register 2. If the computation in M_u halts, then the computation in system Π_u also halts.

The ADD modules and SUB modules in system Π_u are the same as those in Section 4.

In the case of generating number, when the instruction l_h is reached, the system halts (instead of starting to output the computation result as in the case of computing functions). So, the label l_h in $l_{18} : (SUB(4), l_0, l_h)$ can be omitted just halting.

The system Π_u contains

1) 16 neurons for the 8 registers: $\sigma_0, \ldots, \sigma_7, \sigma_{b_0}, \ldots, \sigma_{b_7}$,

2) 19 neurons for the 19 labels (l_h is saved, and three neurons are saved in ADD–ADD and ADD–SUB modules): σ_{l_i} $(0 \leqslant i \leqslant 20, i \neq 6, 10)$,

3) 3 neurons for the 13 SUB instructions: $\sigma_{b_1}, \sigma_{b_2}, \sigma_{b_3}$,

4) 7 neurons in the INPUT-OUTPUT module: $\sigma_{in}, \sigma_{out}, \sigma_{a_1}, \ldots, \sigma_{a_5}$.

Hence, there are 45 neurons in total. Furthermore, each neuron in Π_u has only one rule except for neuron σ_{a_5} with two rules inside.

6 Conclusion and remarks

In this work, the problem of constructing universal WSN P systems with a small number of neurons is investigated. In the systems constructed in this work, the neurons are quite "simple" in the sense that each neuron has only one rule; the system is almost homogeneous in the sense that all neurons have the same set of rules except for one neuron.

The system given in Theorem 2 has 48 neurons and each neuron has only one spiking rule. The system given in Theorem 3 has 45 neurons and each neuron has only one rule except for one neuron with 2 spiking rules. It is possible to use less neurons to construct universal WSN P systems provided that neurons have more spiking rules.

In this work, the parameters, the number of neurons and the number of rules, are considered. Other complexity parameters of universal WSN P systems such as the number of types of weights, the number of synapses, are also worth investigating.

Acknowledgements

The work of Zeng X X and Pan L Q was supported by National Natural Science Foundation of China (Grant Nos. 61033003, 91130034), Ph.D. Programs Foundation of Ministry of Education of China (Grant Nos. 20100142110072, 2012014213008) and National Science Foundation of Hubei Province (Grant No. 2011CDA027). The work of Pérez-Jiménez M J was supported by the project TIN2009-13192 of the Ministerio de Ciencia e Innovación of Spain, cofinanced by FEDER Funds, and the "Proyecto de Excelencia con Investigador de Reconocida Valía" of the Junta de Andalucía under grant P08-TIC04200.

References

1 Wang B, Jin X P, Cheng B. Lion pride optimizer: an optimization algorithm inspired by lion pride behavior. Sci China Inf Sci, 2012, 55: 2369–2389
2 Guo Y, Wang Z X. An immune-theory-based model for monitoring inter-domain routing system. Sci China Inf Sci, 2012, 55: 2358–2368
3 Păun G. Computing with membranes. J Comput Syst Sci, 2000, 61: 108–143
4 Ionescu M, Păun G, Yokomori T. Spiking neural P systems. Fundam Inform, 2006, 71: 279–308
5 Leporati A, Zandron C, Ferretti C, et al. On the computational power of spiking neural P systems. In: Proceedings of the 5th Brainstorming Week on Membrane Computing, Sevilla, 2007. 227–246
6 Wang J, Hoogeboom H J, Pan L, et al. Spiking neural P systems with weights. Neural Comput, 2010, 22: 2615–2646
7 Csuhaj-Varjú E, Margenstern M, Vaszil G, et al. On small universal antiport P systems. Theor Comput Sci, 2007, 372: 152–164
8 Rogozhin Y, Verlan S. On the rule complexity of universal tissue P systems. In: Proceedings of 6th Workshop On Membrane Computing. Berlin: Springer-Verlag, 2006. 356–363
9 Păun A, Păun G. Small universal spiking neural P systems. Biosystems, 2007, 90: 48–60
10 Zhang X, Zeng X, Pan L. Smaller universal spiking neural P systems. Fundam Inform, 2008, 87: 117–136
11 Neary T. A universal spiking neural P system with 11 neurons. In: Proceedings of the 11th International Conference on Membrane Computing, Jena, 2010. 327–346
12 Rozenberg G, Salomaa A. Handbook of Formal Languages. Berlin: Springer-Verlag, 1997
13 Păun G. Membrane Computing: An Introduction. Berlin: Springer-Verlag, 2002
14 Minsky M. Computation—Finite and Infinite Machines. Englewood Cliffs: Prentice Hall, 1967
15 Korec I. Small universal register machines. Theor Comput Sci, 1996, 168: 267–301