Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.12785/amis/080140

Solving Subset Sum Problems by Time-free Spiking Neural P Systems

Tao Song[1,2], *Liang Luo*[1,*], *Juanjuan He*[2], *Zhihua Chen*[2] *and Kai Zhang*[3]

[1] Key Laboratory of High Performance Ship Technology of Ministry of Education, Wuhan University of Technology, Wuhan, China
[2] School of Automation, Huazhong University of Science and Technology, Wuhan 430074, Hubei, China
[3] School of Computer Science, Wuhan University of Science and Technology, Wuhan 430081, Hubei, China

Received: 24 Jun. 2013, Revised: 1 Nov. 2013, Accepted: 3 Nov. 2013
Published online: 1 Jan. 2014

Abstract: In membrane computing, spiking neural P systems (shortly called SN P systems) are a group of neural-like computing models inspired from the way spiking neurons communication in form of spikes. In previous works, SN P systems working in a non-deterministic manner have been used to solve numerical NP-complete problems, such as SAT, vertex cover, in feasible time. In these works, the application of any rule should complete in exactly one time unit, and the precise execution time of rules plays a crucial role on solving the problems in polynomial (or even in linear) time. However, the restriction does not coincide with the biological fact, since in biological systems, bio-chemical reactions may cost different execution time due to the external uncontrollable conditions. In this paper, we consider timed and time-free SN P systems, where the precise execution time of the rules is removed. To investigate the computational efficiency of time-free SN P systems, we solve Subset Sum problem by a family of uniform time-free SN P systems.

Keywords: membrane computing, spiking neural P system, Subset Sum problem, uniform solution, time-freeness.

1 Introduction

Spiking neural P systems (SN P systems, for short) are a class of distributed and parallel computing models inspired by spiking neurons, introduced in [5]. Please refer to a respective chapter of the book [18] for some details of SN P systems, to the membrane computing website [9] for updated information.

With different biological facts, many variants of SN P systems have been proposed, as well as their computational properties have been heavily studied. SN P systems were proved to be computationally complete, i.e., Turing universal, as number generators [5,11,12,13,14,15], language generators [3,21], and function computing devices[16]. SN P systems with cell division or budding can generate new neurons during the computation, thus provide a way to generate exponential working space in polynomial or linear time. These systems were successfully used to (theoretically) solve computationally hard problems, particular in **NP**-hard problems, in a feasible (polynomial or linear) time (see, e.g., [6,7,8,12]).

In the previous obtained SN P systems in [3,5,6,7,8,12,16,20], a global clock is generally assumed marking the time for the system. The systems work in a synchronized manner in the global level (all neurons apply their rules in a parallel manner), and for each neuron, only one of the enabled rules can be used on the tick of the clock. However, programming living things cannot assume general restrictions on execution times. Moreover different biological processes may cost different time due to the external uncontrollable conditions. Therefore, it seems crucial to investigate the power of SN P systems without restrictions on execution times of rules. Time-free cell-like P systems and SN P systems with no restriction on the execution of the rules are investigated in [1] and [10]. The idea of considering solving computational hard problem by time-free P systems was initialled by Matteo in [2], but no specific time-free system constructed for solving a hard computational problem was given in [2].

In this work, we deal with the computational efficiency of time-free SN P systems by solving Subset Sum problems, that is, the correctness of the solution does not depend on the precise timing of the involved processes. Moreover, the family of time-free SN P systems are constructed in a uniform way.

[*] Corresponding author e-mail: luoliang610@163.com

2 Solving Decision Problem by Time-free SN P Systems

In this section, we start by recalling the definition of timed spiking neural P systems [10], and then a particular class of timed SN P systems investigated in this work, called time-free SN P systems, is introduced. The definition is complete, but familiarity with elemental concepts of formal languages and the basic elements of classic SN P systems (e.g. from [17]) is helpful.

A *timed spiking neural P system* of degree $m \geq 1$ is a construct of the form

$$\Pi = (O, \sigma_1, \sigma_2, \ldots, \sigma_m, syn, in, out, e), \text{ where}$$

–$O = \{a\}$ is alphabet of spike (a is called *spike*);
–$\sigma_1, \sigma_2, \ldots, \sigma_m$ are *neurons* with any $\sigma_i = (n_i, R_i)$ $(1 \leq i \leq m)$, where
  (1)$n_i \geq 0$ is the *number of spikes* initially contained in neuron σ_i;
  (2)R_i is a finite set of *rules* : $E/a^c \to a^p$, where E is a regular expression over O, $c \geq 1$ and $c \geq p \geq 0$;
–$syn \subseteq \{1,2,\ldots,m\} \times \{1,2,\ldots,m\}$ with $(i,i) \notin syn$ is a finite set of *synapses* connecting neurons;
–$in, out \in \{1,2,\ldots,m\}$ are *input* and *output* neurons;
–$e : R \to \mathbb{N}$ is a time mapping, where R is the total set of rules holding $R = R_1 \cup R_2 \cdots \cup R_m$.

A rule $E/a^c \to a^p$ with $p \geq 1$ is called *extended spiking rule*; a rule $E/a^c \to a^p$ with $p = 0$ is written in the form $E/a^c \to \lambda$ and is called a *forgetting rule*. The rule of the type $E/a^c \to a$ and $a^c \to \lambda$ is said to be the *standard* spiking and forgetting rule. If $L(E) = \{a^c\}$, then the rules are written in the simplified forms $a^c \to a^p$ and $a^c \to \lambda$

The time mapping $e : R_1 \cup R_2 \cdots \cup R_m \to \mathbb{N}$ specifies the execution time of every rule in the system. We suppose to have an external clock marks time-units of equal length, starting from time 0. In each time unit, a finite number of spikes and a finite number of rules are present in each neuron. The applicability of a rule is determined by checking the total number of spikes contained in the neuron against a regular set associated with the rule. Specifically, the spiking rules are applied as follows. If the neuron σ_i contains exactly k spikes, and $a^k \in L(E), k \geq c$, then the rule $r : E/a^c \to a^p$ is enabled and can be applied. This means consuming (removing) c spikes (thus only $k - c$ spikes remain in neuron σ_i); the neuron is fired, and it produces p spikes after $e(r)$ time units (that is, when the execution of spiking rule terminates). During the execution of the rule, the neuron is in the closed status. If the rule is used at step t and $e(r) = 0$, then the neuron is open at step t; if $e(r) = 1$, then the neuron is closed at step t, and open in the next step, etc. If the rule is used at step t and $e(r) \geq 1$, then at steps $t, t+1, t+2, \ldots, t+e(r)-1$ the neuron is closed, 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) from its neighboring neurons. At the step $t + e(r)$, the neuron becomes open again, so that it can receive spikes (which can be used starting with the step $t + e(r) + 1$, when the neuron can again apply rules). Once emitted from neuron σ_i, the p spikes reach immediately all neurons σ_j such that $(i, j) \in syn$ 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 a neuron contains exactly c spikes, then the forgetting rule $a^c \to \lambda$ is enabled to use. By using the forgetting rule, $c \geq 1$ spikes are removed out of the neuron, thus be removed out of the system. Note that the application of a rule is controlled by the total number of spikes contained in the neuron.

In each time unit, if neuron σ_i has applicable rules, then one of the enabled rules must be used on the tick of the clock, all neurons working in parallel. When a rule from R_i is started to apply in a neuron, then other rules from the neuron can not be applied before the execution completes. It is possible that two spiking rules $E_1/a^{c_1} \to a^{p_1}$ and $E_2/a^{c_2} \to a^{p_2}$ may have $L(E_1) \cap L(E_2) \neq \emptyset$, in this case, at some moment the number of enabled rules might be more than one; only one of them is chosen in a non-deterministic way to use. In any neuron, its rules are used in the sequential manner (at most one in each step), but for different neurons, they work in parallel with each other.

The “state” (also called *configuration*) of the system at any moment is described by the number of spikes in neuron at that moment and the *open-close status* of the neuron, that is, the number of steps to count down until it becomes open again. One can define *transitions* among configurations by using the rules in the neurons. Any sequence of transitions starting from the initial configuration, halting or not, is called a *computation*. The *result of a computation* is defined as the total number of spikes sent into the environment by the output neuron, when the system halts.

The systems can work in a so called “input-output” mode to solve decision problems. An instance of a decision problems can be encoded in form of spike train and introduced in the system through the input neuron(s), and then by performing the computation of the system, a number of spikes are emitted to the environment when the system halts. By reading a instance of the problem into the system, we can obtain the solution of problem, *yes* or *no*, by analyzing the number of spikes emitted to the environment when the computation halts. If the correctness of the solution has no relationship with the time mapping e associated with the system, then the solution is called time-free. In order to formally define the time-free solution to decision problems by using timed SN P systems, we recall the some notions used in SN P systems to solve decision problems.

A decision problem X is a pair (I_X, Θ_X) where I_X is a set of instances of X, and Θ_X is a predicate over I_X. Let $X = (I_X, \Theta_X)$ be a decision problem, if the answer of X is positive, then it is represented as $\Theta_X(u) = 1$; otherwise $\Theta_X(u) = 0$, and $\Pi = \Pi_u, u \in I_X$ be a (countable) family of SN P systems.

–The family of systems Π is *sound*, if for any $u \in I_X$, there exists a computation of Π_u, by which we have $\Theta_X(u) = 1$.
–The family of systems Π is *complete*, if for any $u \in I_X$ such that $\Theta_X(u) = 1$, then we get the positive answer of the instance from every computation of Π_u.
–The family of systems Π is *polynomially bounded* if there exists a polynomial function $p(n)$ such that, for each $u \in I_X$, all computations in Π_u must halt in at most $p(|u|)$ steps.

In timed SN P systems, the execution time of rules is determined by the time mapping e (that is the execution time of rules can be any number), so it is possible that there exits a rule with exponential execution time. In this case, the notion of polynomial bounded cannot be obtained any more. We consider here another way to define the computation time in timed SN P systems, which is called the *rule starting steps* (RS-steps for short). In the computation of the timed SN P systems, only the steps when at least one rule is started to apply are counted. Those steps, in which no rule is started to be executed, are omitted. We can easily expanded the notions sound, complete and polynomial bounded to timed sound, timed complete and timed polynomial bounded, respectively. The notions are given as follows.

Let $\Pi_e = \Pi_u(e), u \in I_X$ be a (countable) family of timed SN P systems.

–The family of systems Π_e is said to be *timed sound*, if for a given time-mapping e, the family $\Pi_e = \Pi_u(e), u \in I_X$ is sound.
–The family of systems Π_e is said to be *timed complete*, if for a given time-mapping e, the family $\Pi_e = \Pi_u(e), u \in I_X$ is complete.
–The family of systems Π is *timed polynomially bounded* if for a given time-mapping e, there exists a polynomial function $p(n)$ such that all computations in any system $\Pi_u(e)$ must halt in, at most, $p(|u|)$ RS-steps.

For any time mapping e, if the family Π_e are timed sound, timed complete and timed polynomially bounded, then the family of systems Π_e are said to be time-free sound, time-free complete and time-free polynomially bounded.

3 A Time-free Uniform Solution to Subset Sum Problem

Subset Sum problem is known NP-complete problem [4], which can be formally defined as follows.

Problem. NAME: SUBSET SUM.

–INSTANCE: a finite $V = \{v_1, v_2, \ldots, v_n\}$ with $v_i \in \mathbb{N}, i = 1, 2, \ldots, n$, and a positive integer number S.
–QUESTION: is there a sub set B of V $(B \subseteq V)$ such that $\sum_{b \in B} = S$?

Theorem 1.*Subset Sum problem can be solved by timed SN P systems in a time-free uniform manner.*

For a given time mapping e, we construct a family of timed SN P systems $\Pi_n, n \in \mathbb{N}$ to solve Subset Sum problems. The systems, shown in Figure 1, are constructed in a uniform way. The input neurons $\sigma_{c_{i,3}}, i = 1, 2, \ldots, n$ (having a synapse pointing to itself) can read spikes from the environment, and the function of the output neuron σ_{out} is to emit spikes to the environment. In the initial configuration, all the neurons contain no spike inside, except that neurons σ_{c_i} and $\sigma_{c_{i,2}}$ $(i = 1, 2, \ldots, n)$ contain 3 spikes and 5 spikes, respectively. For any instance of Subset Sum problem $(V = \{v_1, v_2, \ldots, v_n\}, S)$, $cod(v_i) = 5v_i$ spikes are introduced in each input neuron $\sigma_{c_{i,3}}, i = 1, 2, \ldots, n$ at the beginning of the computation.

In the first step, with 3 spikes inside, neuron $\sigma_i, i = 1, 2, \ldots, n$ can fire for the two rules $a^3 \to a^2$ and $a^3 \to a^3$ are both enabled, but only one of them will be non-deterministically used at that moment. This corresponds to the cases that whether the number v_i is selected in the subset B or not. In neuron σ_i, there are following two cases.

Proof. –If neuron σ_i uses the rule $a^3 \to a^2$, then after certain steps (depending on the execution time of the rule determined by the given e) it emits 2 spikes to neurons $\sigma_{c_{i,1}}$ and $\sigma_{c_{i,2}}$, respectively. Neuron $\sigma_{c_{i,1}}$ removes the 2 spikes by the forgetting rule $a^2 \to \lambda$ and keeps inactive, while neuron $\sigma_{c_{i,2}}$ can fire by the rule $a^7/a^6 \to a^3$, sending 3 spikes to neuron $\sigma_{c_{i,3}}$. By receiving 3 spikes from neuron $\sigma_{c_{i,2}}$, neuron $\sigma_{c_{i,3}}$ gets $5v_i + 3$ spikes and the rule $a^8(a^5)^+/a^8 \to a^6$ is enabled. In some steps, it sends 6 spikes to neurons $\sigma_{c_{i,3}}$ and σ_{out}. Neuron $\sigma_{c_{i,3}}$ contains 7 spikes and will fire again at some step. From that moment on, this process can repeat for $v_i - 1$ times, totally sending $6(v_i - 1)$ spikes to neurons $\sigma_{c_{i,4}}$ and σ_{out}. Note that both of the two neurons cannot fire with $6k, k \in \mathbb{N}$ spikes. When neuron $\sigma_{c_{i,3}}$ has exactly 5 spikes, after receiving 3 spikes from neuron $\sigma_{c_{i,2}}$, rule $a^8 \to a^4$ is enabled. Some steps later, it sends 4 spikes to neurons $\sigma_{c_{i,3}}$, $\sigma c_{i,4}$, and σ_{out}. With $6(v_i - 1) + 4$ spikes, neuron $\sigma_{c_{i,4}}$ can fire, sending 2 spikes to neurons σ_{out} and σ_d. In this case, totally $6v_i$ spikes are emitted to neuron σ_{out} (where the first $6(v_i - 1)$ spikes are from $\sigma_{c_{i,3}}$ and the last 6 spikes are sent by neurons $\sigma_{c_{i,3}}$ and $\sigma_{c_{i,4}}$, which emit 4 and 2 spikes, respectively) and 2 spikes are sent to neuron σ_d.
–If neuron σ_i uses the rule $a^3 \to a^3$, then some steps later it sends 3 spikes to each of neurons $\sigma_{c_{i,1}}$ and $\sigma_{c_{i,2}}$.

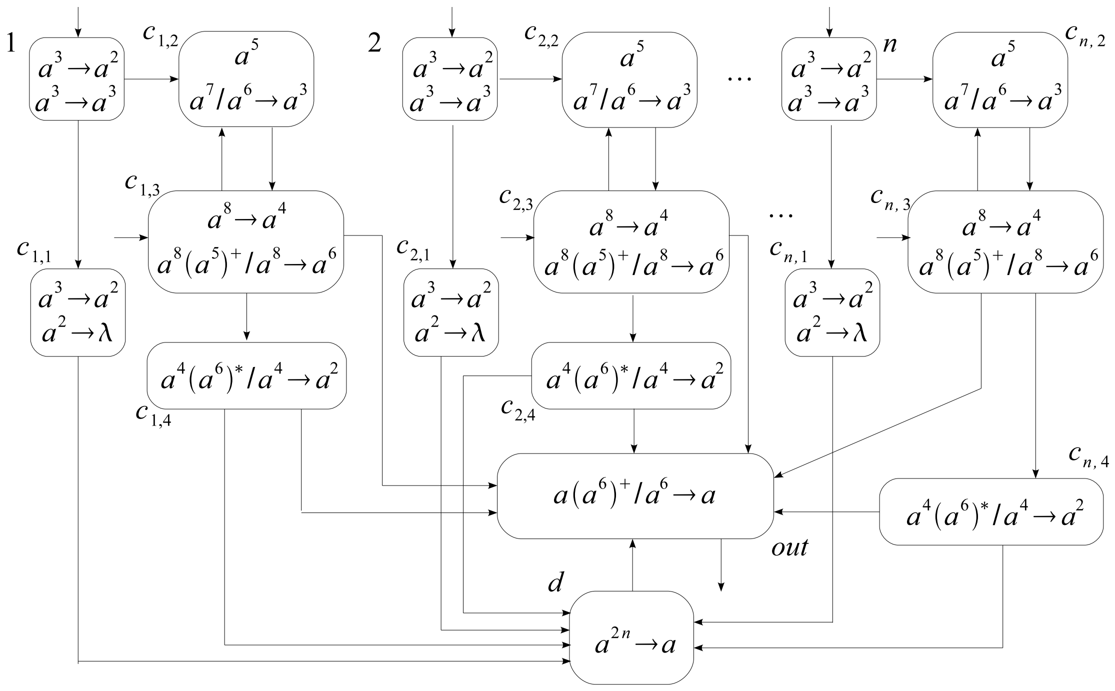

Fig. 1: The timed SN P system Π_n solving Subset Sum problem in a uniform manner

At that time, neuron $\sigma_{c_{i,2}}$ gets 8 spikes and cannot fire for no rule is enabled. Having 3 spikes, neuron $\sigma_{c_{i,1}}$ fires by the rule $a^3 \to a^2$, sending 2 spikes to neuron σ_d at certain step. In this case, no spike is emitted to neuron σ_{out} from the neuron $\sigma_{c_{i,3}}$.

Note that before neuron σ_d is activated, the number of spikes in neuron σ_{out} is always even (it can only receive even number of spikes in any step), so neuron σ_{out} cannot fire until neuron σ_d fires. When neuron σ_d collects $2n$ spikes (it indicates that all the neurons σ_i and $\sigma_{c_{i,j}}, i = 1,2,\ldots,n, j = 1,2,3,4$ have finished their spiking), it becomes activated and sends one spike to neuron σ_{out}. By receiving the spike, the number of spikes in neuron σ_{out} becomes $6n+1$. It can fire one step later by using the rule $a(a)^6/a^6 \to a$ and begin to send spikes to the environment. When the system halts, we can obtain the answer *yes* or *no* to the instance of Subset Sum problem: the answer is positive (*yes*) if exactly S spikes are in the environment; otherwise, the answer is negative (*no*).

We can easily check that system Π_n contains $5n+2$ neurons, and any neuron has at most two spiking/forgetting rules. So, for any system $\Pi_n, n \in N$, there is a deterministic Turing machine constructing it in polynomial time. If the instance of Subset Sum problem has a positive answer, then we can get the positive answer from a computation of Π_n. Dually, if we get the positive answer from a computation of system Π_n, then the instance has a positive answer. Therefore, the system is sound and completeness.

In the following, we will check the RS-steps of the computations in system Π_n. It is possible that the n modules of Π_n (module i consists of neurons $\sigma_i, \sigma_{i,1}, \sigma_{i,2}, \sigma_{i,3}, \sigma_{i,4}, i = 1,2,\ldots,n$) work sequentially, which is the worst case of consuming computation times. According to the description above, in each module i of Π, all the neurons fire in a sequential way. Neuron σ_i only fires once, that is, one RS-step costs in neuron σ_i. If neuron σ_i uses the rule $a^3 \to a^3$, module i finishes its work in two RS-steps. If neuron σ_i uses the rule $a^3 \to a^2$, then neuron $\sigma_{c_{i,2}}$ will fire $v_i - 1$ times, neuron $\sigma_{c_{i_3}}$ will fire v_i times and neuron $\sigma_{c_{i,4}}$ will fire once. So, module i costs at most $1+(v_i-1)+v_i+1 = 2v_i+1$ RS-steps. In the first step, all the neurons σ_i start by using one of their two rules, their spiking costs one RS-step. Hence, the n modules cost at most $\sum_{i=1}^{n} 2v_i + 1$ RS-steps . After all the n modules finish working, neuron σ_d fires once, and neuron σ_{out} maximally fires $\sum_{i=1}^{n} v_i$ times. The system Π_n will halt in at most $\sum_{i=1}^{n} 3v_i + 2$ RS-steps, which is a polynomial time.

It is easy to obtain that the time mapping e has no influence on the result of system Π_n, that is, for any time mapping e, the SN P system Π_n will get the same answer to the same instance of Subset Sum problem. Thus, the family Π_n is time-free sound and time-free complete (with respect to each instance of X_n). For any time mapping e, the family of systems Π_n can halt in at most polynomial RS-steps, hence the systems are time-free polynomially bounded.

Therefore, the SN P system Π_n is a time-free uniform solution to the Subset Sum problem. □

4 Conclusion

In this work, inspired by the fact that different biological biochemistry processes cost different time to complete, we consider timed SN P systems as well as time-free SN P systems, where the execution time of the rules can be different . It is investigated the efficiency of time-free SN P systems. Specifically, a time-free solution of Subset Sum problem is achieved by using a family of uniform time-free SN P systems. It needs to note that in the computation, the notion of rule starting steps (RS-steps) are defined as the computation steps, which can avoid the inherently exponential execution time of the rules in such systems.

Acknowledgment

This work was supported by National Natural Science Foundation of China (61074169, 61202037, 61033003, 91130034, 61100055 and 30870826), Ph.D. Programs Foundation of Ministry of Education of China (20100142110072), Fundamental Research Funds for the Central Universities (2010ZD001 and 2010MS003), and National Science Foundation of Hubei Province (2008CDB113, 2011CDB233 and 2011CDA027).

References

[1] M. Cavaliere, V. Deufemia, Further Results on Time-Free P Systems, International Journal of Foundations of Computer Science, **17**, 1 (2006).
[2] M. Cavaliere, Time-Free Solution to Hard Computational Problems. In the Proceeding of Tenth Brainstorming Week on Membrane Computing, Sevilla, 1 (2012).
[3] H. Chen, M. Ionescu, T.O. Ishdorj, Spiking Neural P Systems with Extended Rules: Universality and Languages, Natural Computing, **7**, 2 (2008).
[4] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to Theory of NP-Completeness, W. H. Freeman and Co., New York (1979).
[5] M. Ionescu, G. Păun, T. Yokomori, Spiking Neural P Systems, Fundamenta Informaticae, **71**, 2-3 (2006).
[6] T.-O. Ishdorj, A. Leporati, L. Pan, X. Zeng, X. Zhang, Deterministic Solutions to QSAT and Q3SAT by Spiking Neural P Systems with Pre-computed Resources, Theoretical Computer Science, **411**, 25 (2010).
[7] A. Leporati, G. Mauri, C. Zandron, G. Păun, M.J. Pérez-Jiménez, Uniform Solutions to SAT and Subset Sum by Spiking Neural P Systems, Natural Computing, **8**, 4 (2009).
[8] A. Leporati, C. Zandron, C. Ferretti, G. Mauri, Solving Numerical NP-Complete Problems with Spiking Neural P Systems, In: G. Eleftherakis, P. Kefalas, G. Păun, G. Rozenberg, A. Salomaa (eds.) Membrane computing, International Workshop, WMC8, Thessaloniki, Greece, (2007).
[9] The P System Web Page: http://ppage.psystems.eu.
[10] L. Pan, X. Zeng, X. Zhang, Time-free Spiking Neural P Systems. Neural Computation, **23**, 5 (2011).
[11] L. Pan, J. Wang, H.J. Hoogeboom, Spiking Neural P Systems with Astrocytes. Neural Computation, 24 (2012).
[12] L. Pan, G. Păun, M.J. Pérez-Jiménez, Spiking Neural P Systems with Neuron Division and Budding. Science China Information Sciences, **54**, 8 (2011).
[13] L. Pan, X. Zeng, X. Zhang, Y. Jiang, Spiking Neural P Systems with Weighted Synapses. Neural Processing Letters, **35**, 1 (2012).
[14] L. Pan, X. Zeng, Small Universal Spiking Neural P Systems Working in Exhaustive Mode. IEEE Transactions on Nanobioscience, **10**, 2 (2011).
[15] L. Pan, G. Păun, Spiking Neural P Systems: An Improved Normal Form. Theoretical Computer Science, **411**, 6 (2010).
[16] A. Păun, G. Păun, Small universal spiking neural P systems. BioSystems, **90**, 48-60, 2007.
[17] G. Păun, M.J. Pérez-Jiménez, Spiking Neural P Systems: An overview, In: A.B. Porto, A. Pazos, W. Buno (Eds.), Advancing Artificial Intelligence through Biological Process Applications, PA: Medical Information Science Reference, Hershey, (2008).
[18] G. Păun, G. Rozenberg, A. Salomaa (Eds.), Handbook of Membrane Computing, Oxford University Press, (2010).
[19] G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, Berlin: Springer-Verlag, **3**, (1997).
[20] J. Wang, H.J. Hoogeboom, L. Pan, G. Păun, M.J. Pérez-Jiménez, Spiking Neural P Systems with Weights. Neural Computation, **22**, 10 (2010).
[21] X. Zhang, X. Zeng, L. Pan, On String Languages Generated by Spiking Neural P Systems with Exhaustive Use of Rules. Natural Computing, **7**, 4 (2008).

Tao Song received Ph.D degree from Huazhong University of Science and Technology in 2013. He is now working as a post-doctor in Huazhong University of Science and Technology. His research interests include DNA computing and membrane computing.

Liang Luo received Ph.D. degree from Huazhong University of Science and Technology in 2010. He is a lecturer with Wuhan University of Technology. His current interests include DNA Computing and Digital ship design.

Juanjuan He received Bachelor degree from Huazhong University of Science and Technology in 2008. She is recently a Ph.D student in Huazhong University of Science and Technology. Her research interests include membrane computing, and membrane algorithm.

Zhihua Chen received Ph.D. degree from Huazhong University of Science and Technology in 2009. She is currently an Associated Professor at Huazhong University of Science and Technology. Her research interests include DNA computing and information security.

Kai Zhang received Ph.D. degree from Huazhong University of Science and Technology in 2008. He is an Associate Professor at Wuhan University of Science and Technology. His research interests include graph theory, molecular computation and intelligent computing.