Spiking Neural P Systems with Structural Plasticity: Attacking the `Subset Sum` Problem

Francis George C. Cabarle[1(✉)], Nestine Hope S. Hernandez[1], and Miguel Ángel Martínez-del-Amor[2]

[1] Algorithms and Complexity Lab, Department of Computer Science, University of the Philippines Diliman, Diliman, 1101 Quezon City, Philippines
fccabarle@up.edu.ph, nshernandez@dcs.upd.edu.ph
[2] Department of Computer Science and AI, University of Sevilla, Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
mdelamor@us.es

Abstract. Spiking neural P systems with structural plasticity (in short, SNPSP systems) are models of computations inspired by the function and structure of biological neurons. In SNPSP systems, neurons can create or delete synapses using *plasticity rules*. We report two *families of solutions*: a *non-uniform* and a *uniform* one, to the **NP**-complete problem `Subset Sum` using SNPSP systems. Instead of the usual rule-level nondeterminism (choosing which rule to apply) we use synapse-level nondeterminism (choosing which synapses to create or delete). The nondeterminism due to plasticity rules have the following improvements from a previous solution: in our non-uniform solution, plasticity rules allowed for a normal form to be used (i.e. without forgetting rules or rules with delays, system is simple, only synapse-level nondeterminism); in our uniform solution the number of neurons and the computation steps are reduced.

Keywords: Membrane computing · Spiking neural P system · Structural plasticity · **NP**-complete · `Subset Sum`

1 Introduction

Membrane computing, [18] a branch of *natural computing*, aims to abstract and obtain computing ideas, data structures and operations from the function and structure of living cells. Several introductory and advanced books [8,19] (including a handbook [21]) have been produced for this branch, as well as a recent collection of applications to systems and synthetic biology [9]. As early as 2006, *membrane algorithms* [16] have been introduced for approximation inspired by *P systems* (the model of computations in membrane computing). The P systems webpage [1] includes an updated list of workshops, conferences, and books on or related to membrane computing (including a collection of PhD theses). The Thomson Reuters Institute for Scientific Information (in short, ISI) has identified

membrane computing as a "fast emerging research front" as early as October 2003, see e.g. [2].

In this work, the specific P systems we consider are *spiking neural P systems* (in short, SNP systems) first introduced in [12]. In particular, we focus on a variant of SNP systems known as *spiking neural P systems with structural plasticity* (in short, SNPSP systems), recently introduced in [4] and improved and extended in [7]. We do not go into the details of SNP systems here, including their neuroscience inspirations, computing power (i.e. what a model can or cannot compute) and computational complexity (i.e. time and space efficiency in solving problems). We refer the reader instead to good introductions in [12,20] and the SNP systems chapter in the membrane computing handbook [21]. In SNPSP systems, *neurons* are placed on the vertices of a directed graph, and the edges between neurons are called *synapses*. Aside from *spiking rules* (more details below) which are used to consume and produce spikes, SNPSP systems have *plasticity rules*. Plasticity rules allow a neuron σ_i to create or delete synapses from itself (i.e. outgoing edges of σ_i) but cannot create or delete synapses towards itself (incoming edges of σ_i). The plasticity rules in SNPSP systems are inspired by actual structural plasticity in biological neurons [3].

In this work we use SNPSP systems to provide *families of solutions* to the **NP**-complete problem `Subset Sum`. The hardness of the `Subset Sum` problem is applied to practical and important use in order to secure many systems requiring encryption, e.g. in [11]. Of course, when we refer to solutions to a problem, we mean to say that we provide an *algorithm* solving the problem, where the algorithm in this case is the constructed SNPSP system.

This paper is organized as follows: some preliminaries for the rest of this work are given in Sect. 2. Syntax and semantics of SNPSP systems in Sect. 3. The `Subset Sum` problem as well as some existing solutions using SNP systems are provided in Sect. 4. Complexity classes of SNPSP systems, with respect to the type of solution, are also provided in Sect. 4. A non-uniform family of solutions is given in Sect. 5. A uniform family of solution is provided in Sect. 6. Lastly, Sect. 7 provides some final remarks and future research directions.

2 Preliminaries

Before proceeding to the syntax (i.e. elements that constitute a model) and semantics (i.e. the meaning and use of elements of a model) of SNPSP systems, we briefly recall *regular expressions*. Regular expressions will be used by neurons to check which spiking or plasticity rules to apply. We denote the set of natural (counting) numbers as $\mathbb{N} = \{0, 1, 2, \ldots\}$, where $\mathbb{N}^+ = \mathbb{N} - \{0\}$. Let V be an alphabet, V^* is the set of all *finite* strings over V with respect to *concatenation* and the *identity element* λ (the empty string). The set of all non-empty strings over V is denoted as V^+, so $V^+ = V^* - \{\lambda\}$.

A language $L \subseteq V^*$ is *regular* if there is a regular expression E over V such that $L(E) = L$. A regular expression over an alphabet V is constructed starting from λ and the symbols of V using the operations union, concatenation, and $+$.

Specifically, (i) λ and each $a \in V$ are regular expressions, (ii) if E_1 and E_2 are regular expressions over V then $(E_1 \cup E_2)$, E_1E_2, and E_1^+ are regular expressions over V, and (iii) nothing else is a regular expression over V. With each expression E we associate a *language* $L(E)$ defined in the following way: (i) $L(\lambda) = \{\lambda\}$ and $L(a) = \{a\}$ for all $a \in V$, (ii) $L(E_1 \cup E_2) = L(E_1) \cup L(E_2)$, $L(E_1E_2) = L(E_1)L(E_2)$, and $L(E_1^+) = L(E_1)^+$, for all regular expressions E_1, E_2 over V. Unnecessary parentheses are omitted when writing regular expressions. If $V = \{a\}$, we simply write a^* and a^+ instead of $\{a\}^*$ and $\{a\}^+$. If $a \in V$, we write $a^0 = \lambda$.

3 Spiking Neural P Systems with Structural Plasticity

In this section we define SNP systems with structural plasticity. Motivations and recent results in SNPSP systems are included in a series of papers in [5–7]. A spiking neural P system with structural plasticity (SNPSP systems) of degree $m \geq 1$ is a construct of the form $\Pi = (O, \sigma_1, \ldots, \sigma_m, syn, out)$, where:

- $O = \{a\}$ is the singleton alphabet (a is called spike);
- $\sigma_1, \ldots, \sigma_m$ are neurons of the form $(n_i, R_i), 1 \leq i \leq m$; $n_i \geq 0$ indicates the initial number of spikes in σ_i; R_i is a finite rule set of σ_i with two forms:
  1. Spiking rule: $E/a^c \rightarrow a$, where E is a regular expression over O, $c \geq 1$;
  2. Plasticity rule: $E/a^c \rightarrow \alpha k(i, N)$, where E is a regular expression over O, $c \geq 1$, $\alpha \in \{+, -, \pm, \mp\}$, $k \geq 1$, and $N \subseteq \{1, \ldots, m\} - \{i\}$;
- $syn \subseteq \{1, \ldots, m\} \times \{1, \ldots, m\}$, with $(i, i) \notin syn$ for $1 \leq i \leq m$ (synapses between neurons);
- $in, out \in \{1, \ldots, m\}$ indicate the input and output neuron labels.

Given neuron σ_i (we also say neuron i or simply σ_i) we denote the set of neuron labels with σ_i as their presynaptic (postsynaptic, respectively) neuron as $pres(i)$, i.e. $pres(i) = \{j | (i, j) \in syn\}$ (as $pos(i) = \{j | (j, i) \in syn\}$, respectively). Essentially, $|pres(i)|$ and $|pos(i)|$ is the out- and in-degree of the neuron (i.e. vertex) σ_i, respectively. Spiking rules are applied as follows: If neuron σ_i contains b spikes and $a^b \in L(E)$, with $b \geq c$, then a rule $E/a^c \rightarrow a \in R_i$ can be applied. Applying such a rule means consuming c spikes from σ_i, thus only $b - c$ spikes remain in σ_i. Neuron i sends one spike to every neuron with a label in $pres(i)$ at the same step as rule application. A *writing convention* we adopt is as follows: if a rule $E/a^c \rightarrow a$ has $L(E) = \{a^c\}$, we simply write this as $a^c \rightarrow a$.

Plasticity rules are applied as follows. If at step t we have that σ_i has $b \geq c$ spikes and $a^b \in L(E)$, a rule $E/a^c \rightarrow \alpha k(i, N) \in R_i$ can be applied. The set N is a collection of neurons to which σ_i can create a synapse to, or remove a synapse from, using the applied plasticity rule. The rule consumes c spikes and performs one of the following, depending on α:

- If $\alpha := +$ and $N - pres(i) = \emptyset$, or if $\alpha := -$ and $pres(i) = \emptyset$, then there is nothing more to do, i.e. c spikes are consumed but no synapses are created or removed. The former case corresponds to the case when σ_i has a synapse to all neurons with labels in N, while the latter corresponds to the case when σ_i has no more outgoing synapses to delete.

– for $\alpha := +$, if $|N - pres(i)| \leq k$, deterministically create a synapse to every σ_l, $l \in N_j - pres(i)$. If however $|N - pres(i)| > k$, *nondeterministically* select k neurons in $N - pres(i)$, and create one synapse to each selected neuron.
– for $\alpha := -$, if $|pres(i)| \leq k$, deterministically delete all synapses in $pres(i)$. If however $|pres(i)| > k$, *nondeterministically* select k neurons in $pres(i)$, and delete each synapse to the selected neurons.

If $\alpha := \pm$ ($\alpha := \mp$, respectively), create (delete, respectively) synapses at step t and then delete (create, respectively) synapses at step $t+1$. Only the application priority of synapse creation or deletion is changed, but the semantics of synapse creation and deletion remain the same as when $\alpha \in \{+, -\}$. Neuron i can receive spikes from t until $t + 1$, but σ_i can only apply another rule at time $t + 2$.

An important note is that for σ_i applying a rule with $\alpha \in \{+, \pm, \mp\}$, creating a synapse always involves a sending of one spike when σ_i connects to a neuron. This single spike is sent at the time the synapse creation is applied, i.e. whenever synapse (i, j) is created between σ_i and σ_j during synapse creation, we have σ_i immediately transferring one spike to σ_j.

SNPSP systems are *locally sequential* (at each step, at most one rule is applied per neuron) but *globally parallel* (neurons operate in parallel). Note that the application of rules in neurons are *synchronized*, i.e. a global clock is assumed and if a neuron can apply a rule then it must do so. A *configuration* of an SNPSP system is based on (a) distribution of spikes in neurons, and (b) neuron connections based on *syn*. For some step t, we can represent: (a) as $\langle s_1, \ldots, s_m \rangle$ where $s_i, 1 \leq i \leq m$, is the number of spikes contained in σ_i; for (b) we can derive $pres(i)$ and $pos(i)$ from *syn*, for a given σ_i. The initial configuration therefore is represented as $\langle n_1, \ldots, n_m \rangle$, with the possibility of a disconnected graph.

Rule application (as defined above) allows for *transitions* from one configuration to another. A *computation* is defined as a *sequence of transitions*, from an initial configuration, and following rule application semantics. A computation halts if the system reaches a halting configuration, i.e. a configuration where no rules can be applied. The *output* neuron applying a rule (we also say firing) triggers an output of the system, which will be defined below. The output neuron sends spikes to the *environment*, and $pres(out) = \emptyset$. The *input* neuron receives spikes from the environment and $pos(in) = \emptyset$.

An example of an SNPSP system is $\Pi = (\{a\}, \sigma_l, \sigma_m, \sigma_n, syn, n)$ where $\sigma_l = (1, \{a \to \pm 1(l, \{m, n\})\})$, $\sigma_m = \sigma_n = (0, \{a \to a\})$, $syn = \{(m, n)\}$, and the output neuron is σ_n. However and in what follows, for the sake of brevity we omit formally defining the SNPSP system construct. We instead provide a graphical representation as in Fig. 1. A computation of Π is as follows: the initial

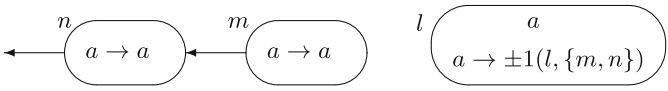

Fig. 1. An SNPSP system Π.

configuration is $\langle 1,0,0\rangle$ representing the corresponding spike distribution in the neuron order $\sigma_l, \sigma_m, \sigma_n$.

Since $a \in L(a)$, the plasticity rule of σ_l can be applied (we denote this as step t). Since $\alpha := \pm$ and $k = 1 < |\{m,n\}|$, σ_l must nondeterministically choose to create either synapse (l,m) or (l,n) at t. At t therefore we have either the synapse set $syn' = syn \cup \{(l,m)\}$ or $syn'' = syn \cup \{(l,n)\}$. Also at t we have σ_l immediately sending a spike to σ_m if (l,m) is created, and the spike distribution is now $\langle 0,1,0\rangle$. If (l,n) is created, the spike distribution is $\langle 0,0,1\rangle$, since σ_n immediately receives a spike from σ_i during synapse creation. At step $t+1$ the created synapse at t is deleted, since $\alpha := \pm$, so we have $syn = \{(m,n)\}$ again. Notice that if (l,m) was created, the single spike is sent out to the environment (i.e. σ_n spikes) at step $t+2$. Otherwise the spike is sent to the environment at $t+1$ (if (l,n) was created).

4 Solving `Subset Sum` with SNPSP systems

SNP systems have been used to solve many **NP**-complete problems, see e.g. [13–15,17,22]. These solutions are usually categorized as either *non-uniform* or *uniform* solutions. A problem Q is solved in a non-uniform way if for each specified instance I of Q we build an SNPSP system $\Pi_{Q,I}$, whose structure and initial configuration depend on I. Furthermore, $\Pi_{Q,I}$ halts and the output neuron spikes at a specified time interval if and only if I is a positive instance of Q. A uniform solution to Q consists of a family $\{\Pi_Q(n)\}_{n\in\mathbb{N}}$ of SNPSP systems such that, given an instance $I \in Q$ of size n, we introduced a polynomial (in n) number of spikes in specified (set of) input neuron(s) of $\Pi_Q(n)$. Again, $\Pi_Q(n)$ halts and the output neuron spikes at a specified time, if and only if I is a positive instance.

More formally, let $X = (I_X, \Theta_X)$ be a decision problem, and $g : \mathbb{N} \to \mathbb{N}$ a computable function, where I_X is a set of instances and Θ_X is a predicate over I_X. We say X is solvable by a family $\mathbf{\Pi} = \{\Pi(n) | n \in \mathbb{N}\}$ of SNPSP systems, in time bounded by g, in a *nondeterministic* and *uniform way* (denoted as $X \in \mathbf{NSNP}(g)$) if the following hold:

- The family $\mathbf{\Pi}$ is polynomially uniform by Turing machines, i.e. there exists a deterministic Turing machine working in polynomial time which constructs $\Pi(n), n \in \mathbb{N}$.
- There exist polynomial time computable functions, *cod* and s, over I_X, such that
  - For each instance $w \in I_X$, $s(w)$ is a natural number, and $cod(w)$ is a valid input (using some encoding) of the SNPSP system $\Pi(s(w))$.
  - The family $\mathbf{\Pi}$ is g-bounded with respect to (X, cod, s), i.e. for each instance $w \in I_X$, the minimum length of an accepting computation of $\Pi(s(w))$ with input $cod(w)$ is bounded by $g(|w|)$.
  - The family $\mathbf{\Pi}$ is *sound* with respect to (X, cod, s), i.e. for each $w \in I_X$, if there exists an accepting computation of $\Pi(s(w))$ with input $cod(w)$, then $\Theta_X(w) = 1$.

- The family $\mathbf{\Pi}$ is *complete* with respect to (X, cod, s), i.e. for every $w \in I_X$, if $\Theta_X(w) = 1$ then there exists a computation of $\Pi(w)$ with input $cod(w)$ which is an accepting computation.

We say $X = (I_X, \Theta_X)$ is solvable in polynomial time by a family $\mathbf{\Pi} = \{\Pi(n) | n \in \mathbb{N}\}$ of SNPSP systems, in a nondeterministic and uniform way (denoted as $X \in \mathbf{NPSNP}$) if there exists a $k \in \mathbb{N}$ such that X is solvable by the family $\mathbf{\Pi}$ in time bounded by a polynomial, in a nondeterministic and uniform way.

The preference of uniform solutions over non-uniform solutions is given by the fact that the former are more strictly related to the structure of the problem, instead of specific instances of the problem. For non-uniform solutions, input neurons are not needed since the problem instance is embedded in the system structure (e.g. number of spikes, neurons, or rules) while in uniform solutions, at least one input neuron is needed to introduce the instance into the system.

Deterministic and nondeterministic solutions (both for non-uniform and uniform solutions) can be found in [13–15,17,22]. Note that nondeterministic solutions allow for more "compact" solutions, in terms of the number of neurons in the system. Unless $\mathbf{P} = \mathbf{NP}$, we need exponential space (i.e. neurons) to deterministically solve hard problems in polynomial time.

The **NP**-complete problem considered here, `Subset sum`, can be defined as follows:
Problem: `Subset Sum` [10]

- Instance: S, and a (multi)set $V = \{v_1, v_2, \ldots, v_n\}$, with $S, v_i \in \mathbb{N}$ and $1 \leq i \leq n$;
- Question: Is there a sub(multi)set $B \subseteq V$ such that $\sum_{b \in B} b = S$?

In [15], the `Subset sum` problem was also solved in a nondeterministic and non-uniform way using SNP systems with *extended rules*: extended rules, as compared to spiking rules, are of the form $E/a^c \rightarrow a^p$ with the meaning that each step a neuron can produce $p \geq 1$ spikes instead of only one spike. Additionally, [15] used some neurons that applied rules sequentially, while some neurons applied their rules in an *exhaustive* manner (i.e. it is possible to apply a rule more than once in one step). A follow-up and improved (uniform) solution was then given in [14]. There are several ways of encoding an instance of `Subset Sum` as the input to the system. Two common ways (used in this work, and as [15] and [14]) involve either (`i`) starting with an initial configuration where each σ_i stores v_i number of spikes, $1 \leq i \leq n$ (for the non-uniform solution), or (`ii`) each σ_{in_i} receives a number of spikes from the environment equal to non-zero multiples of $v_i, 1 \leq i \leq n$ (for the uniform solution).

5 A Non-uniform Solution to `Subset Sum`

We begin by providing a family $\mathbf{\Pi}$ of nondeterministic and non-uniform SNPSP systems solving `Subset Sum` in constant time, as given in [7]. Actually, the

non-uniform solution provided in this section fixes a "bug" in the non-uniform solution given in [7]. The non-uniform solution provided in [7] indeed solves the `Subset Sum` problem, but it is possible for the solution to produce false positive results. The size of each $\Pi \in \mathbf{\Pi}$ is dependent on the value of n and each $v_i, 1 \leq i \leq n$. The number of neurons is a function of the magnitude of each v_i.

The construct of each SNPSP system $\Pi_{\mathtt{ss},I}, \Pi_{\mathtt{ss}} \in \mathbf{\Pi}$, that solves instance I of the `Subset Sum` problem is as follows:

$$\Pi_{\mathtt{ss},I} = \big(\{a\}, \big\{\sigma_i, \sigma_{i_{(Y)}}, \sigma_{i_{(N)}}, \sigma_{i_{(j)}}, \sigma_{out} \big| 1 \leq i \leq n, 1 \leq j \leq v_i\big\}, syn, out\big)$$

where $V = \{v_1, v_2, \ldots, v_n\}$ and we need to check for the value S. We refer to Fig. 2 for a graphical representation of $\Pi_{\mathtt{ss}}$. Compared to the non-uniform solution in [15], each $\Pi_{\mathtt{ss}}$ has the following *normal form* (i.e. a simplifying set of restrictions): (`i`) *simple*, i.e. each neuron has exactly one rule; (`ii`) only *synapse-level nondeterminism*, i.e. nondeterministic choice exists only in choosing which synapse to create and not which rule to apply (known as *rule-level nondeterminism*; (`iii`) no forgetting rule or rule with a delay is used.

The initial configuration is where every σ_i has one spike, and every other neuron has none. In step 1, each neuron σ_i nondeterministically chooses to create either synapse $\big(i, i_{(Y)}\big)$ or $\big(i, i_{(N)}\big)$. If $\big(i, i_{(N)}\big)$ is created, neuron $\sigma_{i_{(N)}}$ consumes a spike but has $pres\big(i_{(N)}\big) = \emptyset$, hence no more computations can proceed. If $\big(i, i_{(Y)}\big)$ is created, then at step 2, $\sigma_{i_{(Y)}}$ then it sends one spike each to $\sigma_{i_{(1)}}$ to $\sigma_{i_{(v_i)}}$, i.e. v_i number of spikes are produced since $\big|pres\big(i_{(Y)}\big)\big| = v_i$.

Once neurons $\sigma_{i_{(1)}}$ to $\sigma_{i_{(v_i)}}$ receive one spike each, they send one spike each to σ_{out} at step 3. If exactly S number of spikes are received by σ_{out} then σ_{out} will send a spike to the environment. Therefore if an affirmative answer to the problem instance exists, a spike would be sent to the environment in four steps since the initial configuration. Otherwise, no spike sent to the environment in four steps indicates a negative answer to the instance. Whether σ_{out} sends a spike or not, the system still halts in four steps. This ends the description of the non-uniform solution.

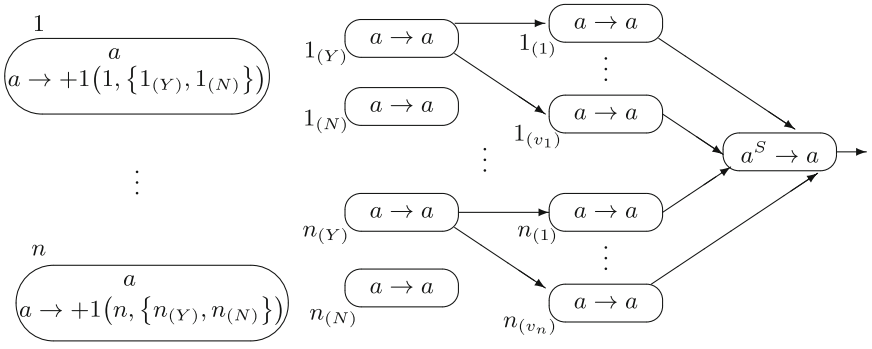

Fig. 2. The non-uniform SNPSP system $\Pi_{\mathtt{ss}}$ solving `Subset Sum`.

In $\Pi_{\texttt{ss}}$ the computation time is constant (four steps) but the number of neurons is dependent on the individual values of the input numbers $v_i, 1 \leq i \leq n$. In this case, our non-uniform solution using exactly one (plasticity) rule in some neurons is enough to replace the functions of rules with delays and forgetting rules in the non-uniform solution in [15]. At the price of extending the computation time, we obtain a uniform solution in the following section.

6 A Uniform Solution to `Subset Sum`

Next, we provide a family $\mathbf{\Pi}$ solving `Subset Sum` in a nondeterministic and uniform way in constant time. In this case, the system only "knows" the value of n, while S and each $v_i, 1 \leq i \leq n$ must be introduced into the system using $n+1$ input neurons. This uniform family of solutions improves the solution provided in the previous section and the uniform solution in [15].

Each SNPSP system $\Pi_{\texttt{us}}(n), \Pi_{\texttt{us}} \in \mathbf{\Pi}$, that solves instance I of size n of the problem is illustrated in Fig. 3. We introduce $2v_i, 1 \leq i \leq n$ spikes into the corresponding input neuron in_i, while we introduce $2S$ spikes into in_{n+1}. Figure 3 shows these spikes already present in the $n+1$ input neurons. These even number of spikes cannot be used by σ_{in_i} until a spike is received from σ_{c_i}. Neurons $\sigma_{c_i}, 1 \leq i \leq n$ are the only neurons with nondeterminism (synapse-level). These neurons nondeterministically allow their corresponding σ_{in_i} neurons to spike (if σ_{c_i} creates synapse (c_i, in_i)) or not (if σ_{c_i} creates synapse (c_i, x)). Note that there is no σ_x so that (c_i, x) is never actually created.

Neurons σ_{c_i} apply their rules at step 1, and at step 3 neurons $e_{i,1}$ spike if they become activated from their corresponding in_i neuron. It also takes 3 steps before σ_{h_3} and σ_{h_4} begin to spike, starting with the spiking of σ_{h_1} at step 1. Neurons σ_{h_3} and σ_{h_4} "feed" a spike to each other starting at step 3. A spike is also sent from σ_{h_4} to σ_{t_1} (the "comparison trigger" neuron). At step 4 the odd number of spikes in σ_{t_1}, sent by the activated $e_{i,1}$ neurons and σ_{h_4}), allows the use of its forgetting rule to remove all of its spikes.

At step 5 only σ_{h_4} sends a spike to σ_{t_1}. At step 6, σ_{t_1} sends one spike each to σ_{h_4} and σ_{t_2}, while σ_{t_1} receives one more spike from σ_{h_4}. At step 7, σ_{h_4} stores two spikes so it can never spike again, while σ_{t_1} sends one more spike to σ_{t_2}. At step 8, σ_{t_2} sends a spike to σ_{acc} and $\sigma_{n_{n+1}}$ which become activated with an odd number of spikes now. Both σ_{acc} and $\sigma_{n_{n+1}}$ empty their spikes (removing two spikes each step) while sending one spike each to σ_{out}.

If the number of spikes accumulated in σ_{acc} equals the $2S$ number of spikes in $\sigma_{n_{n+1}}$, then the system will halt without producing a spike to the environment. Otherwise, σ_{out} will receive one spike from either σ_{acc} or $\sigma_{n_{n+1}}$ and send one spike to the environment. Halting without σ_{out} producing a spike, and halting with σ_{out} producing a spike, corresponds to a positive and negative answer to the problem instance, respectively. This ends the description of the uniform solution.

The number of neurons is constant, with $4n + 9$ neurons. The system halts in at most $2\sum_{i=1}^{n} v_i + 6$ steps: we have one initial step; at most $max\{v_i | 1 \leq i \leq n\} + 1$ to move the spikes from σ_{in_i} to σ_{t_1}; one step for σ_{t_1} to send its first (out

of two) spike; σ_{acc} and $\sigma_{in_{n+1}}$ become activated after two steps, once σ_{t_1} has sent two spikes; at most $\sum_{i=1}^{n} v_i$ steps for comparison between the spikes of σ_{acc} and $\sigma_{in_{n+1}}$; the last step is for σ_{out} to send one spike to the environment. Since $max\{v_i | 1 \leq i \leq n\} \leq \sum_{i=1}^{n} v_i$, we obtain the upper bound for the halting time.

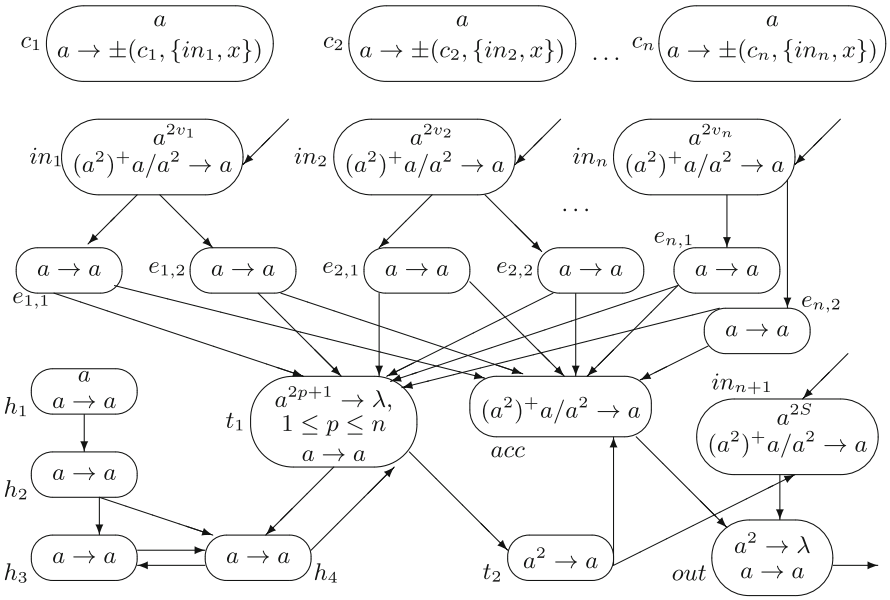

Fig. 3. The uniform SNPSP system $\Pi_{\texttt{us}}$ solving `Subset Sum`.

Actually, the forgetting rules in $\Pi_{\texttt{us}}$ can be removed by using a plasticity rule that functions like a forgetting rule, which is done by the σ_{c_i} neurons (synapse (c_i, x) is never created). The number of neurons and the halting time will still remain the same. In comparison, the non-uniform system in [15] solving `Subset Sum` (using forgetting rules, rules with delays, and standard rules) computes in four steps, while their uniform solution halts in at most $3\sum_{i=1}^{n} v_i + 6$ steps using $5n + 13$ neurons (also using delays, forgetting rules, and standard rules). Thus, one benefit of using synapse-level nondeterminism in this case is decreasing the needed neurons by a linear amount. Also in this case, fewer number of neurons helped improve the computation time: spikes have fewer neurons to pass through, so the spike of σ_{out} is sent to the environment sooner rather than later.

7 Final Remarks

In this work, we fixed the non-uniform solution to `Subset Sum` in [7] using SNPSP systems. We also provided a uniform family of solutions to `Subset Sum`

using SNPSP systems. The use of plasticity rules in this case allowed for a simplifying set of requirements (i.e. normal form) to be applied to our non-uniform solution, compared to the non-uniform solution in [15]. In particular, in our non-uniform solution the plasticity rules could replace forgetting rules and rules with delays. Our uniform solution decreased the number of neurons compared to a uniform solution to `Subset Sum` using SNP systems in [15]. A clear research direction of interest is to show how to solve other hard problems using plasticity rules. Also, how do we make better use of the nondeterminism at the synapse level due to plasticity rules, to perhaps encode problem instances?

Synapse-level nondeterminism in this work provided a reduction in the number of neurons, but perhaps we can also use plasticity rules to further reduce system parameters, e.g. number of neurons, number of rules in neurons, or synapses in the system. As an extension and future work, we also plan to use SNPSP systems to solve other hard problems, in particular, combinatorial problems come to mind: since plasticity rules can (non)deterministically try to create connections (i.e. synapses), one natural use of such rules is to try different combinations of connections in order to solve problems. These work extensions will provide further complexity classes (e.g. semi- or non-uniform) and members of these classes.

Acknowledgements. F.G.C. Cabarle is grateful for the support of the HRIDD HRDP grant I-15-0626-06 of the DOST PCIEERD, Philippines. N. H. S. Hernandez is supported by the UPAA San Francisco & Mely & Rick Ray foundation professorial chair, and the HRIDD HRDP grant I-15-1006-19 of the DOST PCIEERD, Philippines. M.A. Martínez-del-Amor acknowledges the support of the Alain Bensoussan Fellowship programme of ERCIM, and of the project TIN2012-37434 of the "Ministerio de Economía y Competitividad" of Spain, co-financed by FEDER funds.

References

1. P systems web page. http://ppage.psystems.eu/
2. ISI emerging research front, October 2003. http://esi-topics.com/erf/october2003.html
3. Butz, M., Wörgötter, F., van Ooyen, A.: Activity-dependent structural plasticity. Brain Res. Rev. **60**(2), 287–305 (2009)
4. Cabarle, F.G.C., Adorna, H.N., Ibo, N.: Spiking neural P systems with structural plasticity. In: Proceedings of Asian Conference on Membrane Computing (ACMC), Chengdu, China, 4–7 November 2013 (2013)
5. Cabarle, F.G.C., Adorna, H.N., Pérez-Jiménez, M.J.: Asynchronous spiking neural P systems with structural plasticity. In: Calude, C.S., Dinneen, M.J. (eds.) UCNC 2015. LNCS, vol. 9252, pp. 132–143. Springer, Heidelberg (2015)
6. Cabarle, F.G.C., Adorna, H.N., Pérez-Jiménez, M.J.: Sequential spiking neural P systems with structural plasticity based on max/min spike number. Neural Comput. Appl., 1–11 (2015)
7. Cabarle, F.G.C., Adorna, H.N., Pérez-Jiménez, M.J., Song, T.: Spiking neural P systems with structural plasticity. Neural Comput. Appl. **26**(8), 1905–1917 (2015)

8. Ciobanu, G., Păun, G., Pérez-Jiménez, M.J. (eds.): Applications of Membrane Computing. Springer, Heidelberg (2006)
9. Frisco, P., Gheorghe, M., Pérez-Jiménez, M.J. (eds.): Applications of Membrane Computing in Systems and Synthetic Biology. Springer, Heidelberg (2014)
10. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York (1979)
11. Impagliazzo, R., Naor, M.: Efficient cryptographic schemes provably as secure as subset sum. J. Cryptology **9**(4), 199–216 (1996)
12. Ionescu, M., Păun, G., Yokomori, T.: Spiking neural P systems. Fundamenta Informaticae **71**(2–3), 279–308 (2006)
13. Leporati, A., Gutiérrez-Naranjo, M.A.: Solving subset sum by spiking neural P systems with pre-computed resources. Fundamenta Informaticae **87**, 61–77 (2008)
14. Leporati, A., Mauri, G., Zandron, C., Păun, G., Pérez-Jiménez, M.J.: Uniform solutions to sat and subset sum by spiking neural P systems. Nat. Comput. **8**(4), 681–702 (2009)
15. Leporati, A., Zandron, C., Ferretti, C., Mauri, G.: Solving numerical NP-complete problems with spiking neural P systems. In: Eleftherakis, G., Kefalas, P., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2007. LNCS, vol. 4860, pp. 336–352. Springer, Heidelberg (2007)
16. Nishida, T.Y.: Membrane algorithms. In: Freund, R., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2005. LNCS, vol. 3850, pp. 55–66. Springer, Heidelberg (2006)
17. Pan, L., Păun, G., Pérez-Jiménez, M.J.: Spiking neural P systems with neuron division and budding. Sci. China Inf. Sci. **54**(8), 1596–1607 (2011)
18. Păun, G.: Computing with membranes. J. Comput. Syst. Sci. **61**(1), 108–143 (2000)
19. Păun, G.: Membrane Computing. An Introduction. Springer, Heidelberg (2002)
20. Păun, G., Pérez-Jiménez, M.J.: Spiking neural P systems recent results, research topics. In: Condon, A., Harel, D., Kok, J.N., Salomaa, A., Winfree, E. (eds.) Algorithmic Bioprocesses, pp. 273–291. Springer, Heidelberg (2009)
21. Păun, G., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press, New York (2010)
22. Wang, J., Hoogeboom, H.J., Pan, L.: Spiking neural P systems with neuron division. In: Gheorghe, M., Hinze, T., Păun, G., Rozenberg, G., Salomaa, A. (eds.) CMC 2010. LNCS, vol. 6501, pp. 361–376. Springer, Heidelberg (2010)