Analyses and Implementation of Homogenisation Algorithms for Spiking Neural P Systems in the WebSnapse Tool

Tim Kristian Llanto[1], Joshua Amador[1], Francis George C. Cabarle[1,2], Ren Tristan De La Cruz[1], Daryll Ko[1]

[1]Dept. of Computer Science, University of the Philippines Diliman,
Quezon city, Philippines, 1101
E-mail: {tbllanto,jbamador,fccabarle,radelacruz,dlko}@up.edu.ph
[2]Research Group on Natural Computing,
Department of Computer Science and Artificial Intelligence,
I3US, SCORE lab,
Universidad de Sevilla, Avda. Reina Mercedes s/n, 41012, Sevilla, Spain
E-mail: fcabarle@us.es

1 Introduction

An area of computer science called Membrane Computing (MC) was introduced by Gheorghe Păun in [1]. It is a form of computing that takes inspiration from how living cells work. Its system, called the P system, mimics the structure of a cell and how it communicates with its neighbor cells. This is different from the traditional way of computing since cells can communicate in a distributed and parallel manner [2].

A class of P systems known as Spiking Neural P Systems (in short, SN P systems), use a single symbol representing a spike, commonly represented as the symbol a. Each cell is limited to having a single membrane. This type of cell can be called a neuron [3]. A neuron can be activated and send electrical impulses (called spikes) throughout its axons. Neighboring neurons connected by a synapse to the spiking neuron can receive and accumulate the spikes. Receiving and emitting spikes takes a single time step. This is important since the time interval is an essential variable for the computation of SN P systems. There are different ways we can interpret its spiking train results. One way is to use generative mode, which represents the result of a computation as a set of time intervals between two consecutive spikes in the output. Another way is through accepting mode. In this mode, a number is used as an input on the system. This is represented by the time interval of two input spikes. If the system halts, then the input number is accepted.

SN P systems are shown to be Turing Complete both in generating and accepting modes [4]. From this, multiple attempts to find universality by restricting the system have been made. This includes limiting the maximum number of rules, bounded and unbounded regular expressions, spikes consumed, delay, or synapse out-degree of the graph [5]. This method of restricting SN P systems and still having the same computational power is important in the discussion of normal forms. A popular example of this is the Chomsky normal form where using only two different forms of rules in context-free grammars is enough to generate all context-free sets.

A Homogeneous Spiking Neural P System (HSN P System) is another way of representing SN P systems in a different form but having the same computation. In an HSN P system, each neuron contains the same set of rules [6]. It is also shown in the paper that the universality of HSN P system is retained for such forms. Having the same set of rules allows the important features of the system to be on the structure itself (e.g., the connection of neurons) rather than on an individual rule set.

In order to transform a non-homogenized SN P system to its homogenized counterpart, an algorithm needs to be followed. In the paper [7] they proposed two algorithms on how to homogenize a given SN P system. Both algorithms follow almost the same procedure except for the scaling part. One algorithm uses the Type-2 Subsystem Scaling method while the other uses the Released Spike Scaling method. Algorithm proof is shown in the paper on the correctness of the algorithms. But, the time complexity and empirical running time of the algorithms are yet to be analyzed. In this research, we will attempt to find both the theoretical complexity and the empirical running time of both algorithms. We would also create code implementations of the homogenization procedures using the Python programming language. It is integrated with WebSnapse v2 from [8], to easily visualize both algorithms.

In this paper, the sections are organized as follows: Section 2 discusses the preliminaries for our work. Section 3 discusses works related to the present paper, especially the homogenization algorithms. Our experiments, results, and their discussion are provided in Section 4, which also includes theoretical analyses of the algorithms. Lastly, Section 5 provides conclusions and recommendations for further works.

2 Theoretical Framework

2.1 Formal Language Definitions

This paper assumes that the reader is familiar with automata theory and basic languages. Here, we define some formal language notations that would be helpful for later concepts.

Let V be an alphabet. We denote the empty string as λ. The set of all finite strings of symbols from V is denoted as V^* and the set of all nonempty finite strings is denoted as V^+.

2.2 Spiking Neural P Systems

In membrane computing, Spiking Neural P Systems (SN P system) takes inspirations from the behaviour and structures of a neural cell or neuron. An SN P system consists of a group of nodes called neurons that are connected by directed edges called synapses. A neuron can spike the same way a cell can send electrical pulses to its neighbor cells [3]. Each neuron contains a spike count and a set of rules that determines its behavior. A rule consists of a regular expression, the number of spikes consumed and released when activating the rule, and the delay before the spikes are released. In short, rules dictate the action a neuron is going to do given a specific number of spikes. More formally, the following definition is given in the paper [9] :

Definition 1. *A computing extended SN P system, of degree $m \geq 1$, is a construct of the form*

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

1. $O = \{a\}$ *is the singleton alphabet (a is called spike);*
2. $\sigma_1, \ldots, \sigma_m$ *are neurons, of the form* $\sigma_i = (n_i, R_i), 1 \leq i \leq m$, *where:*
   a) $n_i \geq 0$ *is the initial number of spikes contained in* σ_i;
   b) R_i *is a finite set of rules of the following two forms:*
      i. $E/a^c \rightarrow a^p; d$, *where* E *is a regular expression over* a *and* $c \geq p \geq 1, d \geq 0$;
      ii. $a^s \rightarrow \lambda$, *for* $s \geq 1$, *with the restriction that for each rule* $E/a^c \rightarrow a^p; d$ *of type (i) from* R_i, *we have* $a^s \notin L(E)$;
3. $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ *with* $i \neq j$ *for all* $(i, j) \in syn$, $1 \leq i, j \leq m$ *(synapses between neurons);*
4. $in, out \in \{1, 2, \ldots, m\}$ *indicate the input and the output neurons, respectively.*

Rules that are in the form of 2(b)i are called firing rules (or spiking rules). On the other hand, rules of the form 2(b)ii are called forgetting rules.

Restricting the SN P system so that each firing rule produces only one spike is a class of SN P systems called standard SN P system.

A rule in the form $a^c/a^c \rightarrow a^p; d$ can be written in the simplified form $a^c \rightarrow a^p; d$.

For the firing rule $E/a^c \rightarrow a^p; d$ to work, the number of spikes k in the neuron must be in the regular expression, i.e., $a^k \in L(E)$. The number of spikes consumed c in the rule must also be at most k, i.e., $k \geq c$. If these conditions are met, the rule would be applied. Upon firing, the number of spikes of the neuron would be $k - c$. After d time steps, the neuron would send p spikes to its synapses. While

a rule is waiting to be fired because of the delay, the neuron would close and it cannot receive new incoming spikes during this period. If a spike is sent to this neuron, it is said that the spike is lost. After d steps, the neuron would become open again and sends p spikes to its synapses.

For the forgetting rule $a^s \to \lambda$ to work, the number of spikes k in the neuron must be exactly s, i.e., $k = s$. Upon firing, all k spikes of the neuron would be removed.

It is possible that at least two rules can be applied given the number of spikes k of the neuron. This could happen if $a^k \in L(E_1)$ and $a^k \in L(E_2)$, where E_1 and E_2 are regular expressions of different rules. In this scenario, the computation would non-deterministically choose a rule to be applied from one of them.

Definition 2. *In each time step, the system can be represented by a configuration* $\langle r_1/t_1, \ldots, r_m/t_m \rangle$ *where neuron* σ_i *contains* $r_i \geq 0$ *spikes and will be opened after* $t_i \geq 0$ *steps, for* $i = 1, 2, \ldots, m$. *The initial configuration would be* $C_0 = \langle n_1/0, \ldots, n_m/0 \rangle$ *where* n_i *is the initial number of spikes of neuron* σ_i.

Definition 3. *A neuron and its rules can be classified into these three types according to [10]:*

1. *A bounded neuron has rules of the form* $a^i/a^j \to a^p; d$, *where* $1 \leq j \leq i, p \geq 0$, *and* $d \geq 0$. *These rules are called bounded rules.*
2. *An unbounded neuron has rules of the form* $a^i(a^k)^*/a^j \to a^p; d$, *where* $i \geq 0, k \geq 0$, $i + k \geq 1$, $j \geq 1, p \geq 0$, *and* $d \geq 0$. *These rules are called unbounded rules. Note that bounded rules also classify as unbounded rules.*
3. *A general neuron can contain both bounded and unbounded rules.*

Definition 4. *With these, we can define three types of SN P systems:*

1. *A bounded SN P system has bounded neurons.*
2. *An unbounded SN P system has neurons that are either bounded or unbounded.*
3. *A general SN P system has general neurons (i.e., each neuron has rules that are either bounded or unbounded).*

An SN P system can be represented using a directed graph. Below is an example SN P system with parts shown:

3 Related works

3.1 Normal Forms of SN P Systems

It is shown in [4] that SN P systems are Turing complete. Multiple normal forms of SN P system have been developed over the years to extensively test its universality given some restrictions. In [5], it is shown that having a delay of 0 in each firing rule of an SN P system, having a maximal outdegree of two for each of the synapses, or having firing rules on the simplest form $a^i, i \geq 1$, or a^+ also preserves universality.

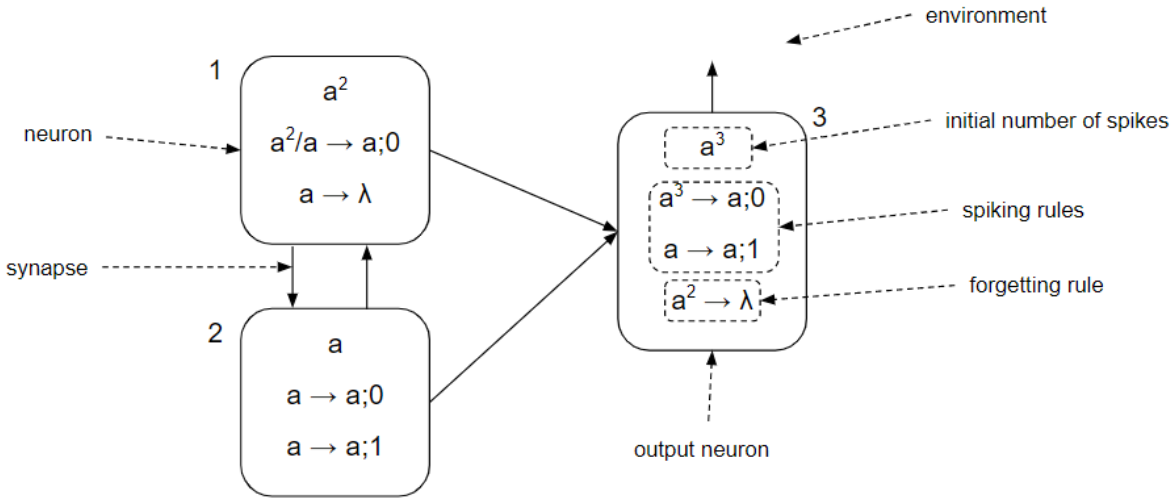

Fig. 1. A graphical representation of an SN P system and its parts

Definition 5. *The following notation is used to present restrictions on an SN P system [11]:*

$$SNP(rule_k, exp_b^{ub}, cons_p, forg_q, dley_r, outd_s)$$

where:

- *k denotes the maximum number of rules each individual neuron can have,*
- *b denotes the number of distinct bounded regular expressions used in all the rules in the entire system,*
- *ub denotes the number of distinct unbounded regular expressions used in all the rules in the entire system,*
- *p denotes the maximum number of spikes consumed by each spiking rule in a single firing,*
- *q denotes the maximum number of spikes consumed by each forgetting rule in a single firing,*
- *r denotes the maximum delay duration, and,*
- *s denotes the maximum synapse out-degree of the graph.*

Below are some examples of normal forms of SN P systems from [11]:

- $N_2SNP(rule_2, exp_0^1, cons_3, forg_0, dley_0, outd_3)$
- $N_{acc}SNP(rule_1, exp_0^1, cons_3, forg_0, dley_0, outd_3)$

N_2SNP here denotes an SN P system in generating mode (i.e., the result of the computation is the set of all time intervals between two consecutive spikes in the output) while $N_{acc}SNP$ denotes an SN P system in accepting mode (i.e., the result of the computation is the set of all input time intervals where the system halts).

In the above normal form for generative mode, having at most two rules per neuron and one type of unbounded regular expression is sufficient for universality. For the accepting mode, the maximum number of rules a neuron can have is reduced to one.

3.2 Homogeneous SN P Systems

In addition to the parameters presented in $SNP(rule_k, exp_b^{ub}, cons_p, forg_q, dley_r, outd_s)$, another restriction would be the limitation of having the same set of rules in each neuron. We call systems with such limitations as Homogeneous Spiking Neural P Systems (HSN P Systems) [6].

It is proven in [6] that HSN P systems are also Turing complete (i.e., $N_2HSNP = NRE$) both in weighted and usual synapses such that only one neuron behaves non-deterministically.

Homogenized neurons can work with many different types of SN P systems and still achieve universality. Some examples include:

- HSN P systems that work in sequential mode as both generating and accepting devices [12].
- Homogeneous SN P systems with anti-spikes (HASN P systems) [13].
- HSN P systems with inhibitory synapses [14].
- Homogeneous SN P systems with structural plasticity (HSNPSP system) [15].

Since the rules in an HSN P system are the same for each neuron, its graphical representation could use a single neuron that contains the homogenized rule set and omit the visual rule set in each neuron. An example of an HSN P system is shown in Figure 2.

Homogenizing an SN P system allows us to focus on the level of the connection of neurons instead of dealing with individual neurons themselves [15].

Spiking Neural P Systems with astrocytes (SNPA systems) uses astrocytes that can inhibit or excite spikes along synapses [16]. The paper [17] uses Homogeneous SNPA (HSNPA) systems to create the Boolean logic gates NOT, OR, AND, NOR, XOR, and NAND. This means that the each neuron contains the same rule set. The rule set that is used contains a single spiking rule $a^*/a \rightarrow a$. Below is an example HSNPA system for the NOR gate:

To simulate a logic gate, an interpretation on what represents 0 and 1 needs to be defined. In this case, an input of a single spike represents the digit 0 and an input of two spikes represents the digit 1. For the output, if the number of total spikes after the system halts is one, then the result would be the digit 0. But if the

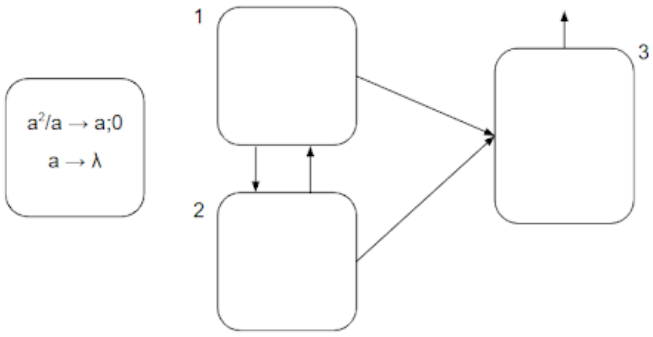

Fig. 2. A graphical representation of an HSN P system

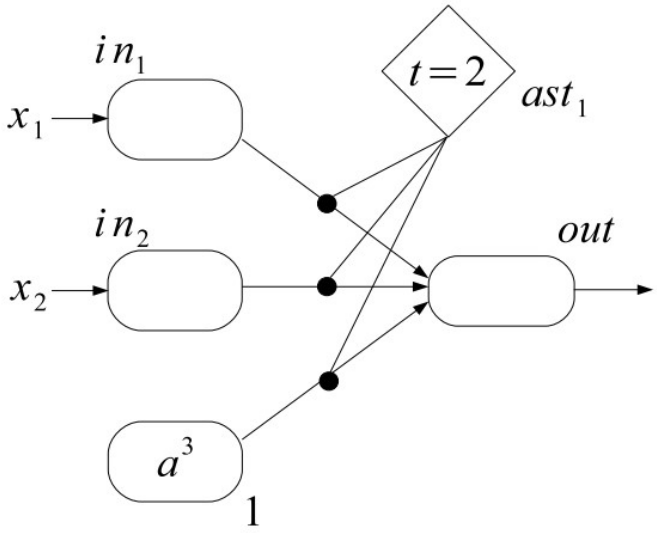

Fig. 3. HSNPA system simulating the NOR boolean logic gate

number of total spikes after the system halts is two, then the result would be the digit 1. For Figure 3, the system outputs two spikes only if both the input neurons receive one spike (i.e., the result would be 1 only if both inputs are 0).

The paper [17] uses the benefits of homogeneous SN P systems to create logic gates. The presented SN P systems are already homogenized and no conversion from a non-homogenized to a homogenized SN P system was shown. In contrast, the present paper implements and analyses the two algorithms presented in [7] that converts a non-homogenized SN P system to a HSN P system.

3.3 Homogenization Algorithms

The paper [7] focused on the creation of two homogenization procedures for SN P systems. We will call the first algorithm as **Algorithm** 1 **:** H **(Type 2 Subsystem Scaling)** and the other as **Algorithm** 2 **:** H' **(Released Spike Scaling)**. Each has its own advantages and limitations. Using any of the two algorithms, a non-homogeneous SN P system Π can be transformed to a corresponding homogeneous SN P system Π' where they compute the same set/languages.

Before discussing the algorithms, some definitions from [7] are first provided. For the homogenization procedure, scaling and translation operations are going to be used for neurons and their rule set. These operations change the number of spikes and/or rules of a neuron. Below are the following operations:

Definition 6. *Neuron translation takes a neuron* $\sigma = (n, R)$ *and a natural number* δ *as input. It produces* $\sigma' = (n+\delta, R')$ *where* $R' = \{a^{\delta}E/a^{c} \rightarrow \beta \mid E/a^{c} \rightarrow \beta \in R\}$.

Definition 7. *Neuron subsystem scaling takes a subsystem sub and a natural number* δ *as input. A subsystem sub of a neuron* σ *is the set of all neurons connected to it. It produces a new subsystem* sub' *where:*

- sub' *contains neuron* $\sigma' = (\delta n, R')$ *where* $R' = \{\delta E/a^{\delta c} \rightarrow \beta \mid E/a^{c} \rightarrow \beta \in R\}$,
- *sub is scaled using either one of Type 1 or Type 2 subsystem scaling.*
  - *(Type 1 subsystem scaling) In sub, if neuron* x *is connected to neuron* σ*, then in* sub' *there will be* δ *copies of neuron* x*. Each copy of neuron* x *will have the same incoming synapses. They would also have the same outgoing synapses to neuron* σ*. Only the original copy will retain the outgoing synapses to other neurons.*
  - *(Type 2 subsystem scaling) In sub, if neuron* x *is connected to neuron* σ*, then in* sub' *there will be* δ *copies of multiplier neurons. Neuron* x *is connected to all the multiplier neurons while all the multiplier neurons are connected to neuron* σ'*. A multiplier neuron would have the rule set of the form* $a^{j} \rightarrow a^{j}$ *where* $j \neq 0$ *is the number of spikes a rule in neuron* x *can produce.*

Algorithm 1 will use Type 2 subsystem scaling. For this to work, the input SN P system must be 1-step delay-tolerant.

Definition 8. *An SN P system is said to be 1-step delay-tolerant if for all* (σ_i, σ_j) *pairs of neurons of system* Π *where* $(i, j) \in syn$ *you can add a delaying neuron* σ_k *between* σ_i *and* σ_j *without changing the set that* Π *computes. The delaying neuron* σ_k *will have rules of the form* $a^{p} \rightarrow a^{p}$ *where* p *represents the number of spikes a rule in* σ_i *can release.*

Below are the homogenization algorithms presented in the paper [7]:

Algorithm 1 `(Type 2 Subsystem Scaling)` H

Input: SN P system Π with set of unique neuron rule sets $\{R_1, ..., R_p\}$.
Output: Homogenized SN P system $\Pi' = H(\Pi)$
$R \leftarrow (pR_1 + 0) \cup (pR_2 + 1) \cup \cdots \cup (pR_p + (p-1))$
Perform type-2 subsystem scaling to all neurons using the factor p
Neurons with rule set $R_i \in \{R_1, ..., R_p\}$ will be translated by $i - 1$
Let R_0 be the union of the rule sets of all multiplier neurons added during type-2 scaling
$t \leftarrow \max\{j \mid a^j \rightarrow a^j \in R_0\} + 1$
$R \leftarrow R_0 \cup (R + t)$
Neuron translate all non-multiplier neurons by t
Use R as the common rule set

Algorithm 1: Type-2 Subsystem Scaling

Below is an example of how Algorithm 1 can be used to homogenize the given SN P system in Figure 4:

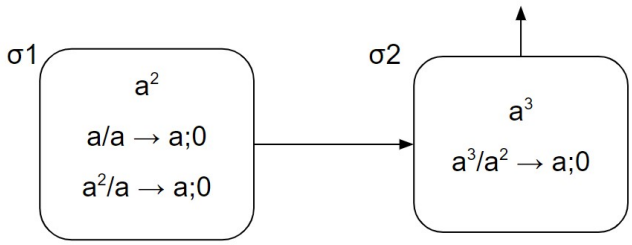

Fig. 4. sample input SN P system Π

The unique neuron rule sets of the input SN P system are $R_1 = \{a^3/a^2 \rightarrow a; 0\}$ and $R_2 = \{a/a \rightarrow a; 0,\ a^2/a \rightarrow a; 0\}$. Since we have two unique rule sets, $p = 2$. We need to find R where $R \leftarrow (2R_1 + 0) \cup (2R_2 + 1)$. Scaling R_1 by 2 and translating it by 0, we get $2R_1 + 0 = \{a^6/a^4 \rightarrow a; 0\}$ while scaling R_2 by 2 and translating it by 1, we get $2R_2 + 1 = \{a^3/a^2 \rightarrow a; 0,\ a^5/a^2 \rightarrow a; 0\}$. Thus $R = \{a^6/a^4 \rightarrow a; 0\} \cup \{a^3/a^2 \rightarrow a; 0,\ a^5/a^2 \rightarrow a; 0\} = \{a^6/a^4 \rightarrow a; 0,\ a^5/a^2 \rightarrow a; 0,\ a^3/a^2 \rightarrow a; 0\}$.

Neuron σ_1 will also contain a modified version of the rule set R_2 and its initial spike count would be scaled by 2 and translated by 1, i.e., σ_1 would contain $2(2) + 1 = 5$ initial spike count. Similarly, neuron σ_2 will also contain a modified version of the rule set R_1 and its initial spike count would be scaled by 2 and translated by 0, i.e., σ_2 would contain $2(3) + 0 = 6$ initial spike count.

Since both neurons are scaled by 2, the spikes received from their incoming edges should also be scaled by 2. Note that in this case, only the edge $\sigma_1 \rightarrow$

σ_2 would be scaled. Two multiplier neurons would be added between these two neurons. Since σ_1 could only produce spike with one count, the rule set inside the multiplier neurons would be of the form $\{a/a \to a; 0\}$, and their initial spike counts set to zero.

Then, we need to find $t \leftarrow \max\{j \mid a^j \to a^j \in R_0\} + 1$ where R_0 is the union of rule sets of all multiplier neurons. In this case, $R_0 = \{a/a \to a; 0\}$. Thus, $j = 1$ and $t \leftarrow 1 + 1 = 2$.

We could now find the common rule set

$$\begin{aligned} R &= R_0 \cup (R + t) \\ &= \{a/a \to a; 0\} \ \cup \ (\{a^6/a^4 \to a; 0,\ a^5/a^2 \to a; 0,\ a^3/a^2 \to a; 0\} + 2) \\ &= \{a/a \to a; 0\} \ \cup \ \{a^8/a^4 \to a; 0,\ a^7/a^2 \to a; 0,\ a^5/a^2 \to a; 0\} \\ &= \{a/a \to a; 0,\ a^8/a^4 \to a; 0,\ a^7/a^2 \to a; 0,\ a^5/a^2 \to a; 0\} \end{aligned}$$

All multiplier neurons must be translated by $t = 2$. So, neuron σ_1 will contain the rule set $R_2+2 = \{a^7/a^2 \to a; 0,\ a^5/a^2 \to a; 0\}$ and its initial spike count would be translated by 2, i.e., σ_1 would contain $5 + 2 = 7$ initial spike count. Similarly, neuron σ_2 will contain the rule set $R_1 + 2 = \{a^8/a^4 \to a; 0\}$ and its initial spike count would be translated by 2, i.e., σ_2 would contain $6+2 = 8$ initial spike count.

Finally, we change the rule set of all neurons to $R = \{a/a \to a; 0,\ a^8/a^4 \to a; 0,\ a^7/a^2 \to a; 0,\ a^5/a^2 \to a; 0\}$.

Thus, the new homogenized SN P system using Algorithm 1 is shown in Figure 5.

Note that this example assumes that the SN P system is 1-step delay-tolerant. When simulated, the input SN P system (Figure 4) outputs the spike train corresponding to the sequence $1, 0, 1$ while the resulting HSN P system (Figure 5) outputs the spike train for the sequence $1, 0, 0, 1$. Thus, this type of homogenization algorithm works if the interpretation of answers is not based on the interval between two spikes.

Algorithm 2: Released Spike Scaling

Alternatively, we can use Algorithm 2 to homogenize the given input SN P system in Figure 4.

In this case, the algorithm uses the same steps except that we need to scale the output spikes of each neuron instead of creating multiplier neurons. Thus, $2R_1 + 0 = \{a^6/a^4 \to 2a; 0\}$ and $2R_2 + 1 = \{a^3/a^2 \to 2a; 0,\ a^5/a^2 \to 2a; 0\}$ and $R = \{a^6/a^4 \to 2a; 0,\ a^3/a^2 \to 2a; 0,\ a^5/a^2 \to 2a; 0\}$

Neuron σ_1's initial spike count would be scaled by 2 and translated by 1, i.e., σ_1 would contain $2(2) + 1 = 5$ initial spike count. Similarly, neuron σ_2's initial spike count would be scaled by 2 and translated by 0, i.e., σ_2 would contain $2(3)+0 = 6$ initial spike count.

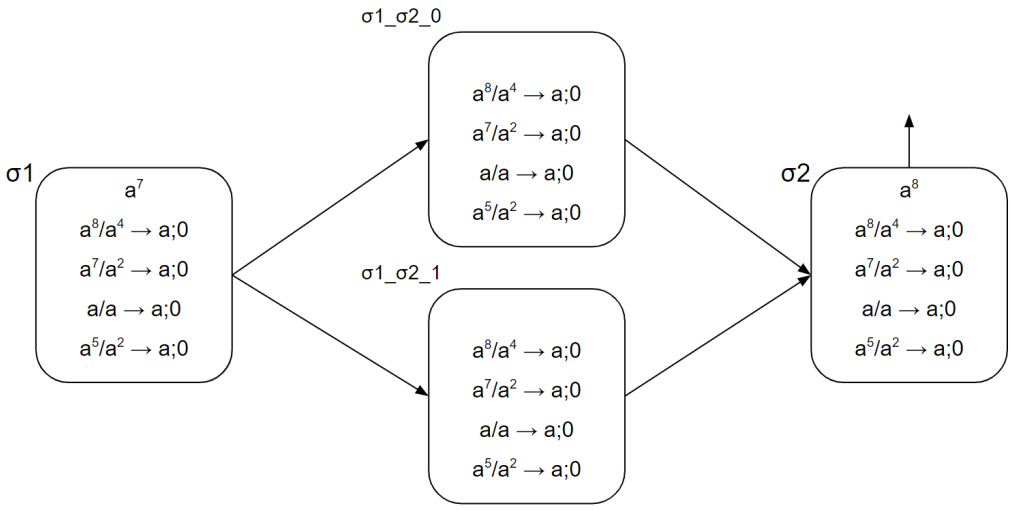

Fig. 5. HSN P system Π' using Algorithm 1

Algorithm 2 (`Released Spike Scaling`) H'

Input: SN P system Π.
Output: Homogenized SN P system $\Pi' = H'(\Pi)$.
Get the set of unique rule sets $\{R_1, ..., R_p\}$.
$R \leftarrow (pR_1 + 0) \cup (pR_2 + 1) \cup \cdots \cup (pR_p + (p-1))$.
Perform scaling to all neurons using the factor p.
Instead of multiplier neurons, scale the number of released spikes by a factor of p.
Neurons with rule set $R_i \in \{R_1, ..., R_p\}$ will be translated by $i - 1$
Use R as the common rule set

Thus, the new homogenized SN P system using Algorithm 2 is shown in Figure 6. Note that this new HSN P system scales the spike train (the output): the spike train corresponds to sequence $2, 0, 2$ instead of $1, 0, 1$.

4 Experiments, Results and Discussion

4.1 Python Code Implementation

The homogenization algorithms are implemented using Python v3.11. A virtual environment is used and the libraries required to run the project can be viewed in requirements.txt file. For easy access to the code, a backend API server is deployed using Flask v2.2.3 and can be accessed in https://homogenize.fly.dev/. Users can make an HTTP POST request to this website to use the homogenization

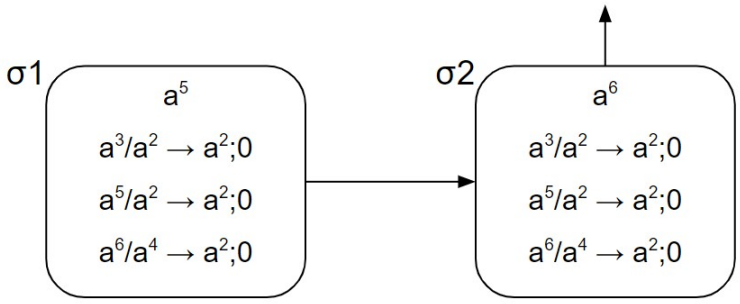

Fig. 6. HSN P system Π'' using Algorithm 2

algorithms. The input SN P system in xmp file format must be sent within the body of the request. Full details on the API request and response structure can be seen on the API website. The links to the source code, documentation, application, examples or test cases, and other information can be accessed (along with other versions of WebSnapse) at the WebSnapse page [18].

4.2 Homogenisation WebSnapse

Homogenisation WebSnapse (HWebSnapse) is a modification of the WebSnapse v2 that contains the homogenize feature. This modification of the WebSnapse v2 was made so that users can now use the homogenization algorithms.

As shown in Figure 7, a "Homogenize" button is presented that users can click. Upon clicking the button, the user must choose between the two homogenization algorithms as shown in Figure 8. Then, the website would send an HTTP POST request to the backend API server containing the SN P system in xmp format. After processing the request, the backend server would send an HTTP response back to the website containing the homogenized form of the SN P system. The website would now update the shown SN P system to its homogenized form. To access the application, the reader can visit the WebSnapse page [18].

4.3 Implementation Testing using WebSnapse v2.

In order to test the functionality of the implemented Python code, trial runs using HWebSnapse were made for multiple examples of SN P systems. Five of these examples were gathered from WebSnapse v2. Additional two test cases were made from the subset sum examples in the paper [19]. The test cases are available at the WebSnapse page [18].

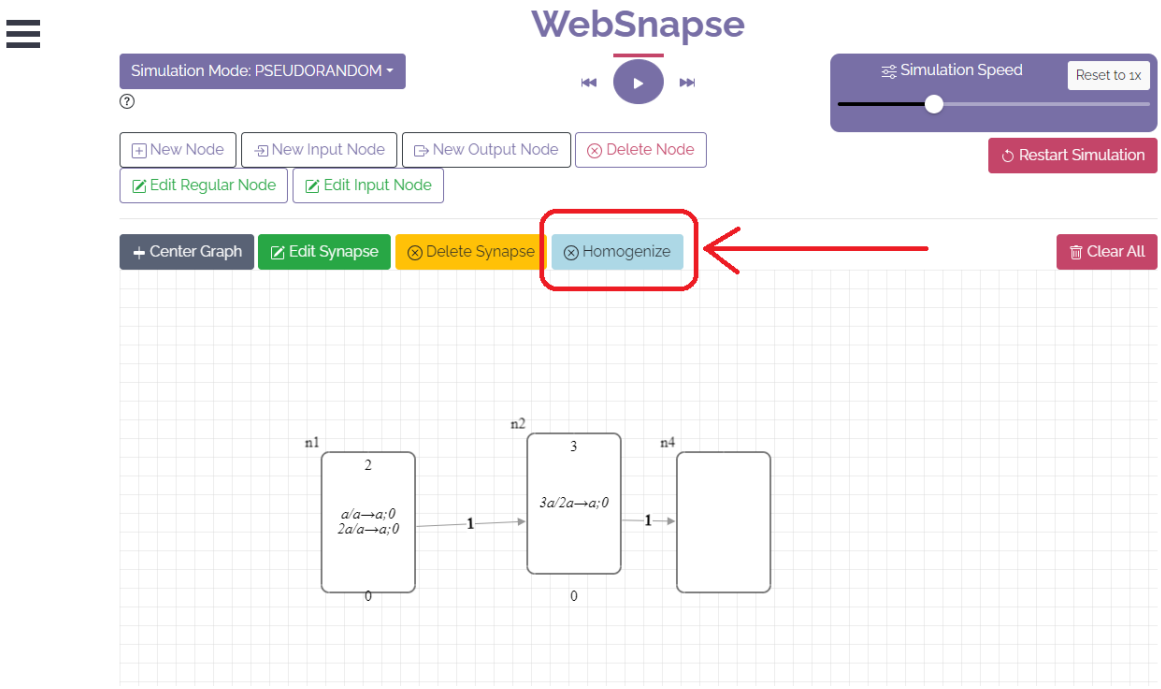

Fig. 7. HWebSnapse showing the Homogenize button

The original heterogeneous SN P system was simulated and their results were recorded. The given SN P system is homogenized using both Algorithm 1 and Algorithm 2. The HSN P systems were simulated and their results were also recorded. The results from the simulations of HSN P Systems were compared to the original heterogeneous SN P system's results. Table 1 shows the summary of the comparison of the results.

The first column shows the different samples of SN P systems used in the test. The first five samples (ex1-ex5) were examples gathered from WebSnapse v2 while the last two were the subset sum examples. The second column shows whether the result from the sample's equivalent HSN P system derived using Algorithm 1 was the same as the original SN P system. Similarly, the third column shows whether the result from the sample's equivalent HSN P system derived using Algorithm 2 was the same as the original SN P system.

For Released Spike Scaling, all HSN P systems produce the expected output.

Output Interpretation

For Type-2 Subsystem Scaling, three samples (those with ✗ marks) cannot use this algorithm since all of them are not 1-step delay-tolerant. Adding multiplier

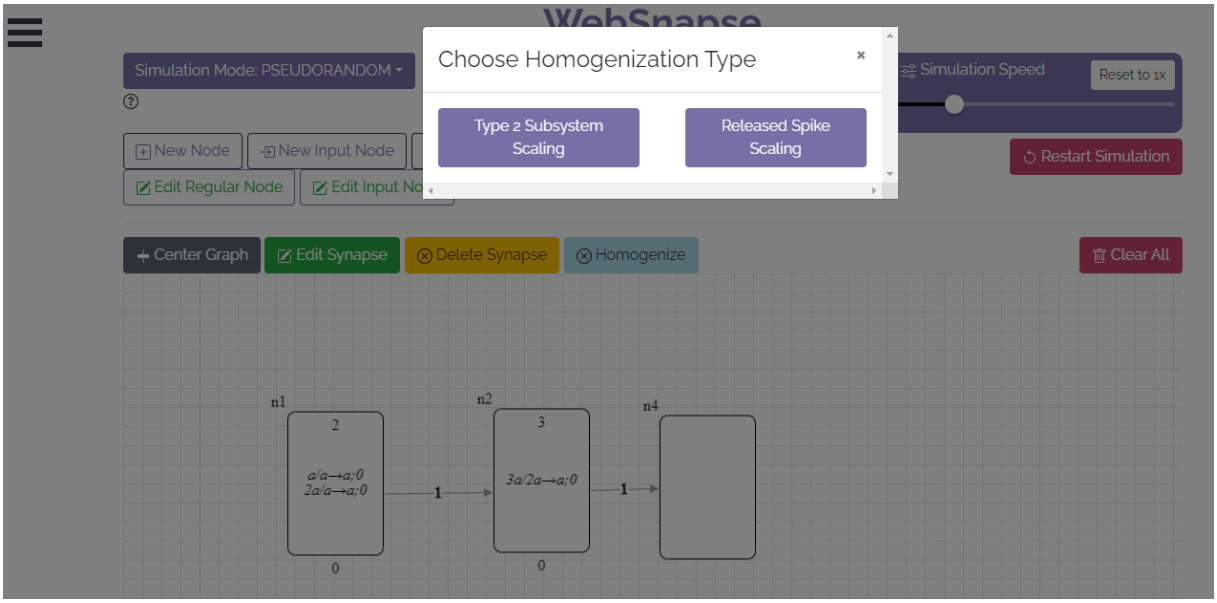

Fig. 8. HWebSnapse showing the choice between the two Homogenization Algorithms

SN P System	Type-2 Subsystem Scaling	Released Spike Scaling
ex1: 3k+3	✗	✓
ex2: Bitadder	✓	✓
ex3: Increasing Comparator	✓	✓
ex4: Even Number Generator	✗	✓
ex5: Number Generator (at least 1)	✗	✓
Subset Sum (Fig 3) ([1,2,4],3)	✓	✓
Subset Sum (Fig 5) ([1,2,3],5)	✓	✓

Table 1. Testing of Homogenization Algorithms using HWebSnapse

neurons changes their expected output. For the other samples, the HSN P system produces the expected output.

From the examples, it is observed that only SN P systems whose result interpretation is based on the interval between spikes are the ones that may not be classified as 1-step delay-tolerant. Furthermore, if the delay occurs before the first spike, then the system is still classified as 1-step delay-tolerant. But, if the delay occurs between the spikes, then the expected output changes, and the SN P system is not classified as 1-step delay-tolerant.

After homogenizing an SN P system that is not 1-step delay-tolerant using Algorithm 1, the time offset between the resulting spikes changes. This behavior changes according to the type of SN P system. For example, a resulting HSN P

system with no loop increases the steps required for the spikes to reach the environment by a linear amount equal to the number of edges it needs to pass through. On the other hand, SN P systems with loops could scale the time offset between the output spikes. An example is the `ex4: Even Number Generator` where its equivalent HSN P system using Algorithm 1 changes its behavior to a multiple of 4 number generator.

For Algorithm 2, the time offset between the output spikes does not change. But, the spikes themselves are scaled by the factor of p.

4.4 Time and Space Complexity Analysis

For the following Big-O complexity notations, we denote n as the number of neurons and k as the maximum number of rules a neuron has.

Algorithm 1: Type-2 Subsystem Scaling

The time complexity of the homogenization algorithm that uses type-2 subsystem scaling is $O(n^3k)$. Consider a complete graph where each neuron is connected to all other neurons. If there are n neurons, there could be a maximum of n unique rule sets. Since the scaling factor p is the same as the number of unique rule sets, i.e. $p = n$, each neuron would be scaled by n. Hence, each edge would contain n new multiplier neurons. Note that in a complete directed graph with n nodes, there would be $n(n-1)$ directed edges. Thus, there would be $n^2(n-1) = n^3 - n^2$ new multiplier neurons in the resulting graph. A multiplier neuron can have a maximum of k rules of the form $a^j/a^j \rightarrow a; 0$. So, the theoretical time it takes to create the multiplier neurons would be $O(k(n^3 - n^2))$ or simply $O(n^3k)$. For the other steps, it takes less than this time complexity to complete.

For its space complexity, it takes up $O(n^4k)$ space. Consider the complete graph scenario defined before. There would be $n^3 - n^2$ multiplier neurons and n original neurons. So, the total number of neurons in the resulting HSN P system would be $n^3 - n^2 + n$. For the common rule set, assume that each neuron contains k rules, with each rule unique from all the other rules. The common rule set would combine all of these rules resulting in nk combined rules for the original neurons. Then, we need to find the maximum number of combined rules the multiplier neurons can have. Since the rule set of a multiplier neuron depends on the rule set of the neuron on its incoming edge, the set of multiplier neurons whose incoming edge came from the same neuron would contain the same rule set. Therefore, there would be a total of nk combined rules for the multiplier neurons. For both the original and multiplier neurons, there would be a total of $nk + nk = 2nk$ combined rules. Since there could be $n^3 - n^2 + n$ neurons and $2nk$ rules in the homogenized common rule set, the total space consumed for the HSN P system would be $O(2nk(n^3 - n^2 + n))$ or simply $O(n^4k)$.

Note that this space complexity assumes that the memory stores the common rule set in each neuron, meaning it has a copy for each neuron. But since the

homogenized rule sets are the same, the implementation could be modified so that it stores only a single copy and each neuron would only reference the same copy. In this kind of implementation, space complexity reduces to $O(n^3 + nk)$.

Algorithm 2: Released Spike Scaling

The time complexity of the homogenization algorithm that uses released spike scaling is $O(nk)$. In this kind of scaling, no new multiplier neuron is introduced. So, the algorithm only iterates over n neurons and for each neuron it iterates over k spikes. Thus, its time complexity would be $O(nk)$.

For its space complexity, it takes up $O(n^2k)$ space. Suppose that each rule is unique from all the other rules. The common rule set would be the combination of the scaled and translated unique rules. So, there would be a total of nk combined and homogenized rules. Note that each of the n neurons would now contain the rule set with nk rules. Thus, the total space required would be the product of the number of neurons and the number of rules in the common rule set, i.e. $n(nk) = n^2k$.

Note that this space complexity assumes that the memory stores the common rule set in each neuron, meaning it has a copy for each neuron. If the implementation assumes that only one copy of the common rule set would be saved, the space complexity reduces to $O(nk + n) = O(nk)$.

Comparison of the Two Algorithms

Figure 9 visualizes the theoretical time complexity of both algorithms when the number of neurons increases while k remains constant. Notice how much slower Algorithm 1 takes to run than Algorithm 2 as the number of neurons increases. The time difference of the two algorithms is in the factor of $n^3k/nk = n^2$.

4.5 Empirical Running Time Analysis

In order to test the accuracy of the theoretical time complexity of the two algorithms, an analysis of the empirical running time was made. Each example from Figure 9 was used as an input for both Algorithm 1 and Algorithm 2 in the Python code implementation and their durations were recorded. Within the Python code, the function `timeit` from the `timeit` library was used to accurately measure the duration of the algorithms. Figure 10 shows the summary of the results. It accurately resembles on that of the theoretical graph from Figure 9.

4.6 Simulation Speed Comparison (Homogenized vs Heterogeneous)

To get the speed comparison of the simulation of the heterogeneous SN P system vs its equivalent homogeneous SN P systems in HWebSnapse, five samples (ex1-ex5) from Table 1 were used. Within the HWebSnapse JavaScript code, the simulation

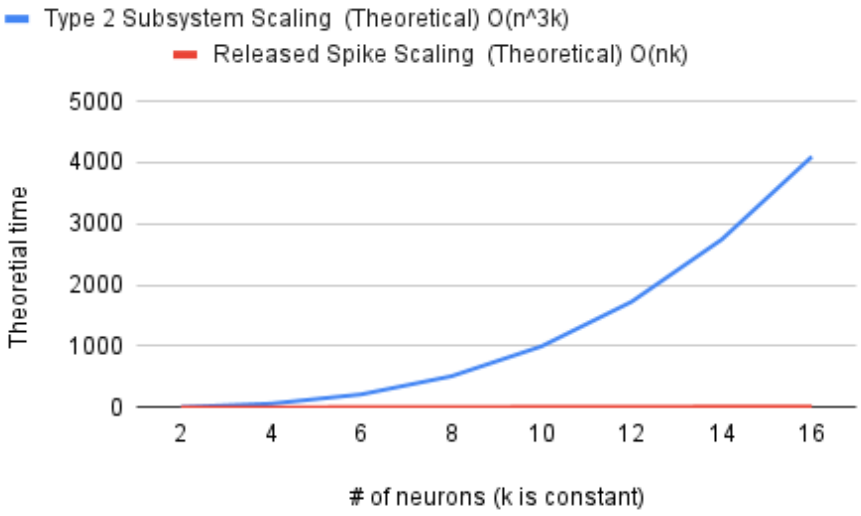

Fig. 9. Comparison of Theoretical Time Complexity of Algorithms 1 and 2

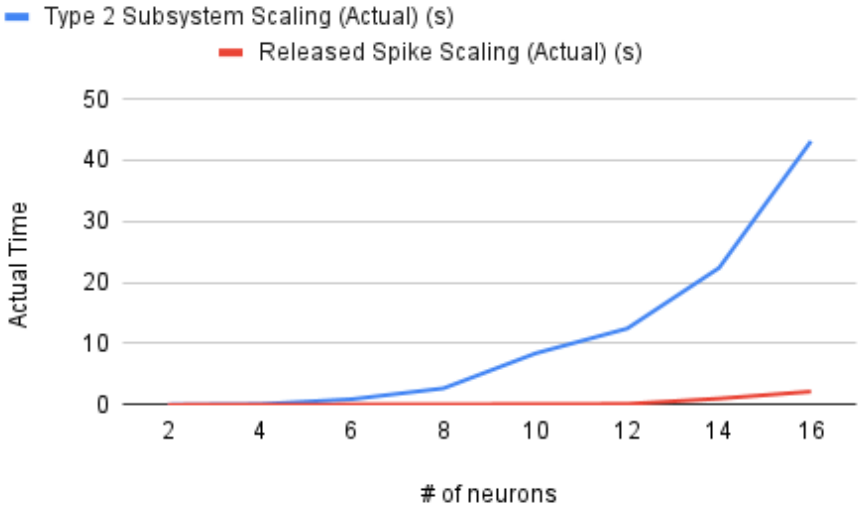

Fig. 10. Comparison of Empirical Running Time of Algorithms 1 and 2

delay speed was removed and the function `performance.now()` was used to calculate the simulation time. For each of the original heterogeneous SN P systems, their simulation time was recorded. Then each would be homogenized using both Algorithm 1 and Algorithm 2. The simulation speeds of these homogenized SN P systems were also recorded. Each simulation of an SN P system was repeated five times and their average was taken. For all the examples, it is evident that the simulation made by Algorithm 2 is much faster than the simulation made by Algorithm 1. Although simulations of HSN P systems produced by both Algorithm 1 and Algorithm 2 were slower than their original heterogeneous counterpart. On average from the five samples, HSN P systems produced by Algorithm 1 were 148% slower than their original heterogeneous counterpart. While HSN P systems produced by Algorithm 2 were only 3% slower.

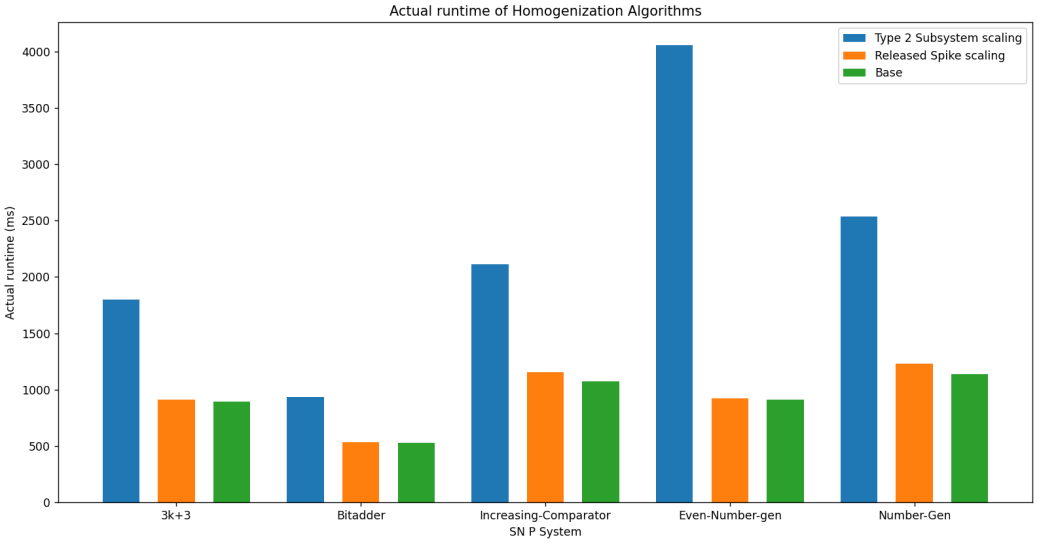

Fig. 11. Simulation Speed of Heterogeneous vs Homogenized SN P Systems

4.7 Type-2 Subsystem Scaling vs Released Spike Scaling

Both algorithms have their own advantages and limitations. Below is a detailed list of their comparison. A summary is shown in Table 2.

1. In terms of the number of neurons, HSN P systems derived using Algorithm 2 have fewer neurons. This is because, in Algorithm 1, multiplier neurons are

added when the scaling operation is used while there are no new neurons added when scaling neurons in Algorithm 2.
2. In terms of algorithmic speed, Algorithm 2 is faster than Algorithm 1 by a factor of n^2 where n is the number of neurons.
3. In terms of memory, Algorithm 2 takes less space than Algorithm 1 by a factor of n^2 where n is the number of neurons.
4. In terms of the simulation speed, simulations of HSN P systems produced by Algorithm 2 are much faster than those produced by Algorithm 1.
5. In terms of the limitation of the type of SN P system input allowed, Algorithm 2 can take any SN P system while Algorithm 1 requires the input to be 1-step delay-tolerant.
6. In terms of flexibility of graph structure, Algorithm 1 can have multiple neuron structures based on the technique of scaling operation used. On the other hand, Algorithm 2 would output the same graph structure.
7. In terms of the input train of spikes, Algorithm 2 requires the spikes to also be scaled by p. For example, the train spikes 0101 where $p = 3$ should be scaled to 0303. On the other hand, Algorithm 1 does not need the input train of spikes to be scaled.
8. In terms of the output train of spikes, Algorithm 2 scales the expected output by p. For example, the output train spikes 0101 where $p = 3$ would be scaled to 0303. On the other hand, Algorithm 1 does not scale the output train spikes.

Advantage	Type-2 Subsystem Scaling	Released Spike Scaling
Smaller number of neurons		✓
Faster algorithm		✓
Less memory		✓
Faster simulation speed		✓
Not limited to 1-Step Delay-Tolerant		✓
Flexibility	✓	
Does not need to scale input	✓	
Does not scale output	✓	

Table 2. Advatanages of Algorithm 1 vs Algorithm 2

4.8 Support for JSON format

Other papers have attempted to improve the overall functionality of WebSnapse v2. One of these is **WebSnapse v3** which improves WebSnapse v2's performance and stability issues [20]. **WebSnapse Reloaded** is another improvement of WebSnapse v2 which enhances its storage, scalability, and maintainability [21].

Both these SN P simulators use the JSON format instead of the XML format that is used in WebSnapse v2. To support these simulators, the backend API of

the homogenization algorithm also allows JSON input and output SN P systems, in addition to XML format.

5 Final Remarks

In the paper [7], two homogenization algorithms were proposed. This study aimed to analyze their time and space complexity, and simulation speeds. An actual code implementation of the algorithms was also made using Python v3.11 and it is integrated as a backend Flask server in HWebSnapse, a spin-off version of the WebSnapse v2 with the homogenization feature. The source codes and test cases are publicly available at [18] and anyone can use them to homogenize a given SN P system following the present paper and based on [7].

As for the topic of homogenization of an SN P system, its uses and advantages are discussed in [6]. If a system is homogenized, this means that each neuron contains the same set of rules. This simplifies the system where its behavior is only determined by the overall structure, i.e., how the neurons are connected, instead of also focusing on rules on each individual neuron. Additionally, the same rule set could also be stored in memory once instead of making a copy for each neuron.

As suggested in the paper [7], future research can focus on possible optimizations of the homogenization algorithms. This could be made easier with the actual Python code implementation. For example, the time complexity of Algorithm 1 that is used on a fully connected graph can be reduced from $O(n^3k)$ to $O(n^2k)$ by considering a different graph design. Consider neuron σ_a with outgoing edges connected to all other neurons $\sigma_{b1}, \sigma_{b2}, ...$ Instead of having a set of multiplier neurons for each outgoing edge of σ_a, we would only have a single set of multiplier neurons and connect them to each of the other neurons. This is possible since the set of multiplier neurons from σ_a to σ_{bi} are the same. Note that it can be further reduced to $O(n^2 + k)$ if the homogenized rule set is only stored once. As for the space complexity of Algorithm 1, using the same set of multiplier neurons reduces the space complexity from $O(n^3 + nk)$ to $O(n^2 + nk)$.

An improvement for the HWebSnapse is instead of a backend server, the homogenization algorithm could be integrated directly into the frontend code, e.g. JavaScript in WebSnapse. This could reduce delay time since computations could be made locally.

It is discussed that Algorithm 1 requires its input to be 1-step delay-tolerant. But, in the paper [7], it is argued that the set of 1-step delay-tolerant SN P systems is also universal. If an algorithm can be made that transforms a non-delay-tolerant SN P system into a delay-tolerant one, the resulting SN P system can be used as an input to Algorithm 1. This would remove the limitation on the input of the algorithm.

Support of the algorithms for newer (and contemporaneous to the present paper) versions such as WebSnapse v3 [20] and WebSnapse Reloaded [21] should be

explored in future studies. These newer versions use JSON format for the representation of their SN P system, instead of the XML format used in WebSnapse v2. The tool in the present paper already supports JSON format, so support for newer versions of WebSnapse, as well as well-known software for P systems such as P-Lingua and MeCoSim [22] has started.

Finally, an analysis of how much memory is reduced when saving HSN P systems is recommended for future research.

Acknowledgements

Support for F.G.C. Cabarle: the Dean Ruben A. Garcia PCA, and Project No. 222211 ORG from the Office of the Vice Chancellor for Research and Development, both from UP Diliman; the *QUAL21 008 USE* project (PAIDI 2020 and FEDER 2014-2020 funds). The authors are thankful to the IMCS Bulletin for allowing us to share our work.

References

1. Gh. Păun, "Computing with membranes," Journal of Computer and System Sciences, vol. 61, no. 1, pp. 108–143, 2000.
2. G. Păun, "A quick introduction to membrane computing," The Journal of Logic and Algebraic Programming, vol. 79, no. 6, pp. 291–294, 2010.
3. M. Ionescu, G. Păun, and T. Yokomori, "Spiking neural p systems," Fundamenta informaticae, vol. 71, no. 2-3, pp. 279–308, 2006.
4. A. Leporati, G. Mauri, and C. Zandron, "Spiking neural p systems: main ideas and results," Natural Computing, pp. 1–21, 2022.
5. O. H. Ibarra, A. Păun, G. Păun, A. Rodríguez-Patón, P. Sosík, and S. Woodworth, "Normal forms for spiking neural p systems," Theoretical Computer Science, vol. 372, no. 2-3, pp. 196–217, 2007.
6. X. Zeng, X. Zhang, and L. Pan, "Homogeneous spiking neural p systems," Fundamenta Informaticae, vol. 97, no. 1-2, pp. 275–294, 2009.
7. R. T. A. de la Cruz, F. G. C. Cabarle, and H. N. Adorna, "Steps toward a homogenization procedure for spiking neural p systems," Theoretical Computer Science, vol. 981, p. 114250, 2024.
8. N. Cruel, C. Quirim, and F. G. C. Cabarle, "Websnapse v2.0: Enhancing and extending the visual and web-based simulator of spiking neural p systems," in Pre-proceedings of the 11th Asian Conference on Membrane Computing, Quezon City, Philippines, pp. 146–166, September 2022.
9. G. Păun, "Spiking neural p systems. a tutorial.," Bulletin of The European Association for Theoretical Computer Science, vol. 91, pp. 145–159, 2007.
10. O. H. Ibarra and S. Woodworth, "Spiking neural p systems: some characterizations," in Fundamentals of Computation Theory: 16th International Symposium, FCT 2007, Budapest, Hungary, August 27-30, 2007. Proceedings 16, pp. 23–37, Springer, 2007.
11. I. C. H. Macababayao, F. G. C. Cabarle, R. T. A. de la Cruz, and X. Zeng, "Normal forms for spiking neural p systems and some of its variants," Information Sciences, vol. 595, pp. 344–363, 2022.

12. K. Jiang, T. Song, W. Chen, and L. Pan, "Homogeneous spiking neural p systems working in sequential mode induced by maximum spike number," International Journal of Computer Mathematics, vol. 90, no. 4, pp. 831–844, 2013.
13. T. Song, X. Wang, Z. Zhang, and Z. Chen, "Homogenous spiking neural p systems with anti-spikes," Neural Computing and Applications, vol. 24, pp. 1833–1841, 2014.
14. T. Song and X. Wang, "Homogenous spiking neural p systems with inhibitory synapses," Neural Processing Letters, vol. 42, 04 2014.
15. R. T. A. de la Cruz, F. G. C. Cabarle, I. C. H. Macababayao, H. N. Adorna, and X. Zeng, "Homogeneous spiking neural p systems with structural plasticity," Journal of Membrane Computing, vol. 3, pp. 10–21, 2021.
16. G. Păun, "Spiking neural p systems with astrocyte-like control.," J. Univers. Comput. Sci., vol. 13, no. 11, pp. 1707–1721, 2007.
17. T. Song, P. Zheng, M. D. Wong, and X. Wang, "Design of logic gates using spiking neural p systems with homogeneous neurons and astrocytes-like control," Information Sciences, vol. 372, pp. 380–391, 2016.
18. "Websnapse page," 2023. `https://aclab.dcs.upd.edu.ph/productions/software/websnapse`.
19. A. Leporati, G. Mauri, C. Zandron, G. Păun, and M. Pérez-Jiménez, "Uniform solutions to sat and subset sum by spiking neural p systems," Natural Computing, vol. 8, pp. 681–702, 12 2009.
20. L. Gallos, J. L. Sotto, F. G. C. Cabarle, and H. N. Adorna, "Websnapse v3: Optimization of the web-based simulator of spiking neural p system using matrix representation, webassembly and other tools," in Pre-proc. 12th Workshop on Computation: Theory and Practice (WCTP2023), 4 to 6 December 2023, Chitose-city, Hokkaido, Japan (S. Hagihara, S. ya Nishizaki, M. Numao, J. Caro, and M. T. Suarez, eds.), pp. 492–510, 2023.
21. M. Gulapa, J. S. Luzada, F. G. C. Cabarle, H. Adorna, K. Buño, and D. Ko, "Websnapse reloaded: The next-generation spiking neural p system visual simulator using client-server architecture," in Pre-proc. 12th Workshop on Computation: Theory and Practice (WCTP2023), 4 to 6 December 2023, Chitose-city, Hokkaido, Japan (S. Hagihara, S. ya Nishizaki, M. Numao, J. Caro, and M. T. Suarez, eds.), pp. 511–536, 2023.
22. L. Valencia-Cabrera, M. Á. Martínez-del Amor, and I. Pérez-Hurtado, "A simulation workflow for membrane computing: From mecosim to pmcgpu through p-lingua," Enjoying Natural Computing: Essays Dedicated to Mario de Jesús Pérez-Jiménez on the Occasion of His 70th Birthday, pp. 291–303, 2018.

6 Appendix

6.1 Appendix A: Test Results of Homogenization Algorithms using WebSnapse v2.

This appendix contains a comparison of the result of the output of seven SN P systems on the result of their homogenized counterparts. The links to the application, the source files, documentation can be found at the WebSnapse page at [18]. The data is in a Spreadsheet file and can be accessed at `https://github.com/pyTimK/Homogeneous-Algorithm-for-SN-P-System/tree/main/analysis/testing`. Analysis of the output spikes are also included within the spreadsheet.

6.2 Appendix B: Comparison of Theoretical Time Complexity of Algorithms 1 and 2

This appendix contains the data on the comparison of the theoretical time complexity of Algorithms 1 and 2. The data is in a Spreadsheet file and can be accessed at `https://github.com/pyTimK/Homogeneous-Algorithm-for-SN-P-System/tree/main/analysis/time_complexity_and_empirical_running_time`. A summary of the spreadsheet can be seen on Figure 9.

6.3 Appendix C: Comparison of Empirical Running Time of Algorithms 1 and 2

This appendix contains the data on the comparison of the empirical running time of Algorithms 1 and 2. The data is in a Spreadsheet file and can be accessed at `https://github.com/pyTimK/Homogeneous-Algorithm-for-SN-P-System/tree/main/analysis/time_complexity_and_empirical_running_time`. A summary of the spreadsheet can be seen on Figure 10.

6.4 Appendix D: Comparison of Simulation Speed of Heterogeneous and Homogenized SN P Systems

This appendix contains data on the comparison of the simulation speed of a heterogenized SN P system and its homogeneous forms. The data is in a Spreadsheet file and can be accessed at `https://github.com/pyTimK/Homogeneous-Algorithm-for-SN-P-System/tree/main/analysis/simulation_speed`. A summary of the spreadsheet can be seen on Figure 11.

6.5 Appendix E: Running The Implemented Python Script

The implementation of the homogenization algorithms is in GitHub and is accessible at the following link: `https://github.com/pyTimK/Homogeneous-Algorithm-for-SN-P-System`. The program can be run as follows:

1. Clone repo using `git clone https://github.com/pyTimK/Homogeneous-Algorithm-for-SN-P-System.git homogeneous_algorithm`
2. Go to the cloned directory
3. Open the command prompt and run `python -m venv venv`
4. Open the virtual environment via `venv\Scripts\activate.bat`
5. Install packages using `pip install -r requirements.txt`
6. Run `python main.py`
7. To use a custom SN P system xmp input, put the file inside `/input` directory and change `input_name` global variable in the `main.py` to the name of the input file.

6.6 Appendix F: Using the Homogenize feature in HWebSnapse

This appendix discusses the step-by-step procedure on how to use the homogenize feature in HWebSnapse.

1. Go to `https://websnapse-homogenize.netlify.app`
2. Create an SN P system from scratch or load an XML file
3. Press the Homogenize Button
4. Select between `Type 2 Subsystem Scaling` or `Released Spike Scaling`

6.7 Appendix G: Test Empirical Running Time of Homogenization Algorithms in Python Implementation

This appendix discusses the step-by-step procedure on how to compute the empirical running time of Algorithm 1 and Algorithm 2 in a given SN P system. This uses the `timeit` library.

1. Open `main.py` in the Python project directory
2. Set the global variable `get_actual_running_time` to `true`
3. Set the global variable `test_runs` to the desired number of test runs. The default is 500
4. Select from any SN P system from the `\input` folder or create one on your own
5. Set the global variable `input_name` to the input name of the SN P system file
6. Run the script using `python main.py`
7. The console would print the average running time for both algorithms

Sample output:

```
ex1.xmp (ScalingType.TYPE_2_SUBSYSTEM_SCALING): 0.022716799983754754 ms
ex1.xmp (ScalingType.RELEASED_SPIKE_SCALING): 0.005746799986809492 ms
```

6.8 Appendix H: Getting the Simulation Time of an SN P system in HWebSnapse

This appendix discusses the step-by-step procedure on how to get the simulation time of an SN P system. A feature in HWebSnapse is included that removes the simulation delay time and prints the execution time on the terminal.

1. Go to `https://websnapse-homogenize.netlify.app/`
2. Create a new SN P system or load an XML file
3. Open the sidebar and tick the checkmark option `Test simulation running time`
4. Close the sidebar and press play
5. The console would print the simulation time in milliseconds.

Sample output:

`The function took 593.5 milliseconds to run with pause times 0.` Note that the pause times when the pause button is pressed is not included in the execution time.