Manchester Metropolitan University

e-space

Manchester Metropolitan University's Research Repository

Please cite the Published Version

Dong, Jianping, Stachowicz, Michael, Zhang, Gexiang, Cavaliere, Matteo, Rong, Haina and Paul, Prithwineel (2021) Automatic Design of Spiking Neural P Systems Based on Genetic Algorithms. International Journal of Unconventional Computing, 16 (2-3). pp. 201-216. ISSN 1548-7199

Publisher: Old City Publishing

Version: Accepted Version

Downloaded from: https://e-space.mmu.ac.uk/627220/

Usage rights: © In Copyright

Additional Information: Author accepted manuscript published by and copyright 2021 Old City Publishing, Inc. Published by license under the OCP Science imprint, a member of the Old City Publishing Group. Email permission received on 11/2/2021.

Enquiries:
If you have questions about this document, contact openresearch@mmu.ac.uk. Please include the URL of the record in e-space. If you believe that your, or a third party's rights have been compromised through this document please see our Take Down policy (available from https://www.mmu.ac.uk/library/using-the-library/policies-and-guidelines)

Automatic Design of Spiking Neural P Systems Based on Genetic Algorithms

JIANPING DONG, MICHAEL STACHOWICZ, GEXIANG ZHANG, MATTEO CAVALIERE, HAINA RONG AND PRITHWINEEL PAUL

Received: January 31, 2020. Accepted: June 10, 2020.

At present, all known spiking neural P systems (SN P systems) are established by manual design rather than automatic design. The method of manual design poses two problems: consuming a lot of computing time and making unnecessary mistakes. In this paper, we propose an automatic design approach for SN P systems by genetic algorithms. More specifically, the regular expressions are changed to achieve the automatic design of SN P systems. In this method, the number of neu-rons in system, the synapse connections between neurons, the num-ber of rules within each neuron and the number of spikes within each neuron are known. A population of SN P systems is created by gen-erating randomly accepted regular expressions. A genetic algorithm is applied to evolve a population of SN P systems toward a successful SN P systems with high accuracy and sensitivity for carrying out specific task. An effective fitness function is designed to evaluate each candi-date SN P system. In addition, the elitism, crossover and mutation are also designed. Finally, experimental results show that the approach can successfully accomplish the automatic design of SN P systems for gen-erating natural numbers and even natural numbers by using the .NET framework.

Keywords: Spiking neural P systems, genetic algorithm, fitness function, mutation probability, membrane computing

1 INTRODUCTION

As a bridge between computer science and natural science, natural computing is a wide research area with several branches [44], which include

cellular automations [15, 21], neural computations [34], elvoving algorithms [14, 26, 36, 41, 42], swarm intelligence [1], artificial immune systems [10], membrane computing [23, 24, 39], etc. Membrane computing has been a new and hot research direction in recent years. Membrane computing models, the thoery of P systems [2, 4, 11, 13, 17–19, 31] or membrane systems, are abstracted from the structure and functioning of the living cell, as well as from the cooperation of cells in tissues, organs and other populations of cells [25, 37]. Currently, membrane systems can be divided, according to different membrane structure, into cell-like P systems, tissue-like P systems and SN P systems [43]. Until now, many variants of membrane computing models have been investigated and also some models have been used in real life application [3, 5, 7, 8, 12, 16, 22, 27, 28, 32, 33].

A good membrane computing model is the basis of its applied investigations and software and hardware implementations. In recent years, with the development of membrane computing models, experts and scholars have started to research the automatic design method of membrane computing models. Currently, the automatic design methods of membrane computing can be classified into two groups [40, 48]: the heuristic algotithms and the reasoning techniques. Genetic algorithms and quantum-inspired evolutionary algorithms are used to evolve a population of P systems [47]. A genetic approach was used to design an artificial cell system, which can be regarded as a cell-like P system with a single membrane [29]. An evolving design solution of membrane systems was proposed to implement the design of square of 4 in the membrane system based on simulation software P-Lingua [9]. A fitness function with a penalty factor was presented to evaluate a P system [30]. Cell-like P systems and tissue-like P systems have been automatically designed to fullfill some specific tasks, like computing square of 4 and of n ($n \geq 2$ is a natural number) [6, 20]. A deterministic and non-halting membrane system by tuning membrane structures, initial objects and evolution rules was proposed in [46]. The reasoning techniques were used to design P systems in [35]. However, this idea has never been extended to the third generation neural networks, which closely model the activity of biological neurons, and more specifically SN P systems [12].

This paper makes the attempt to propose an automatic design approach of SN P systems by genetic algorithms. First of all, we establish a population of SN P systems with changing regular expression on the predefined number of neurons in each SN P system, the synapse connections between neurons, the number of rules within each neuron and the number of spikes within each neuron. Secondly, a genetic algorithm is applied to evolve a population of SN P systems to a successful SN P system with high accuracy and sensitivity for carrying out specific task. An effective fitness function is designed to evaluate each candidate SN P system. Finally, experimental results show that the

approach can successfully accomplish the automatic design of SN P systems for generating natural numbers and even natural numbers by using the .NET framework. The ideas of automated design of membrane computing models (ADMCMs) from this study were given in [38].

The paper is structured as follows. Section 2 describes the problem to solute the automatic design of SN P systems. Section 3 presents the automatic design approach for SN P systems based on genetic algorithms in detail. Experimental results are shown and analyzed in Section 4. Finally, some conclusions are drawn in Section 5.

2 PROBLEM DESCRIPTION

In this section, we briefly review SN P systems, and then the problems of the automatic design of SN P systems are described.

2.1 Spiking Neural P System

A SN P system consists of five main elements: the number of neurons in each SN P system, the synapse connections between neurons, the number of rules in each neurons, the regular expressions which define each rule and the number of spikes within each neuron.

A SN P system [12, 45] of degree $m \geq 1$ is a tuple $\Pi = (O, \sigma_1, \cdots, \sigma_m, syn, i_o)$, where:

1. $O = \{a\}$ is the singleton alphabet (a is called spike);
2. $\sigma_1, \cdots, \sigma_m$ are neurons, identified by pairs

$$\sigma_i = (n_i, R_i), 1 \leq i \leq m \tag{1}$$

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) $E/a^c \rightarrow a; d$ where E is a regular expression over O, and $c \geq 1, d \geq 0$;

(ii) $a^s \rightarrow \lambda$, for some $s \geq 1$, with the restriction that for each rule $E/a^c \rightarrow a; d$ of type (i) from R_i, we have $a^s \notin L(E)$;

3. $syn \subseteq \{1, \ldots m\} \times \{1, \ldots m\}$ with $(i, i) \notin syn$ for $i \in \{1, \ldots m\}$ (synapses between neurons);
4. $i \in \{1, \ldots m\}$ indicates the output neuron (i.e. σ_{i_o} is the output neuron).

2.2 Problem Statement

As briefly introduced above, there are many variants of SN P systems, which are designed by manual design rather than automatic design. In order to automatically generate a SN P system, we should consider each aspect in a SN P system. In this paper, the number of neurons in system, the synapse connections between neurons, the number of rules within each neurons and the number of spikes within each neuron, according to specific task, are previously determined, but the regular expressions which define each rule and the delays on each rule are randomly generated in a SN P system. Then we can generate a population of SN P systems by same method. In our study, the aim is to use genetic algorithms to get an optimal SN P system by approriately evolving a SN P system. The problem is summarized as follows:

Step 1: First of all, we define a population of SN P systems $\Pi = \{\Pi_i\}_{i \in H}$, where H is a subset of natural numbers and each SN P system Π_i of degree $m \geq 1$ is described as follows:

$$\Pi_i = (O, \sigma_1, \cdots, \sigma_m, syn, i_o) \tag{2}$$

where

(a) $O = \{a\}$ is a predefined singleton alphabet;

(b) $\sigma_1, \cdots, \sigma_m$ is the neurons from 1 to m.

$$\sigma_i = (n_i, R_i), 1 \leq i \leq m \tag{3}$$

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) **Spike transfer rules:** $E/a^c \rightarrow a; d$. When fullfilling spike transfer rules and $d = 0$, a spike in the neuron should leave along the synapses and travel to the neurons connected to the neuron where the rule is applied.

(ii) **Spike forgetting rules:** $a^s \rightarrow \lambda$. When performing spike forgetting rules, s spikes are consumed.

Step 2: Determine fitness of each individual in the population.

Step 3: Reserve the individual of the higher fitness parents from the population.

Step 4: Select parents from the population and produce offsprings.

Step 5: Randomly undergo mutation.

Step 6: Check whether any individual meets the requirements, if so, terminate, otherwise continue.
In this study, we want to answer the question: How do we design a SN P systems to perform special task by evolving a initial population of the SN P systems to perform special task.

3 AUTOMATIC DESIGN METHOD

In this section, the detailed procedure of automatically designing a SN P system is described. An overview of automatic design method is outlined and each step is explained one by one.

3.1 Overview of automatical design method

An overview of automatic design method is decribed using two aspects: generating a population of SN P systems and evolving a population of SN P systems. Initial configuration environment is predefined, which includes the number of neurons in each P system, the synapse connections between each neuron, the number of rules and the number of spikes in each neuron. A population of SN P systems is created by randomly generating rules at the start. We evolve the population based on genetic algorithms after generating a population. The pseudocode of automatic design method is shown in Figure 1.

The genenral steps of the design method can be summarized as follows:

Step 1: Input some basic parameters, which include m, n_i, syn, i_o, H, $MaxSteps$, $StepRepetition$, $MutationRate$, $MinFitness$, $MaxGeneration$, $BestFitness$ and $ExpectedSet$,
where:

(a) m, n_i, syn and i_o represent the number of neurons in each P system, the number of spikes in each neuron, the synapse connections between each neuron and the output neurons, respectively.

(b) H is population size.

(c) $MaxSteps$ represents the maximum steps that each network will take.

(d) $StepRepetition$ is the amount of repetitions each network will undergo to generate an output list.

(e) $MutationRate$ is the percentage chance for mutation.

(f) $MinFitness$ represents minimal fitness.

(g) $MaxGeneration$ is the max amount of generations.

Input: Initial membrane construction and objects and genetic algorithm
1: i=1
2: **while** ($i \leq H$) **do**
3: Generating random $SNPS_i$
4: Caculating fitness value F($SNPS_i$)
5: **if** ($F(SNPS_i) \leq MinFitness || F(SNPS_i) == null$) **then**
6: Generating new $SNPS_i$ and replacing old $SNPS_i$
7: **end if**
8: $i = i + 1$
9: **end while**
10: **while** ($generation \leq MaxGeneration$) **do**
11: Caculating fitness value each SNPS
12: Sorting population accordding to set F($SNPS$)
13: i=1
14: **while** ($i \leq H$) **do**
15: **if** ($i \leq Elitism$ && $i \leq H$) **then**
16: $Newpopulation[i] = Population[i]$
17: **if** (F($SNPS_i > BestFitness$)) **then**
18: $BestFitness = SNPS_i$
19: **end if**
20: **else**
21: Parent1=ChooseParent( )
22: Parent2=ChooseParent( )
23: Child=Crossover(Parent1,Parent2)
24: Child=Mutate(Child)
25: $Newpopulation[i] = Child$
26: **end if**
27: **if** ($F(SNPS_i) == 0 || F(SNPS_i) == null$) **then**
28: $F(SNPS_i) = 0$
29: **else**
30: $F(SNPS_i) = F(SNPS_i)$
31: **end if**
32: **end while**
33: $genration = gneration + 1$
34: **end while**
Output: Spiking neural P system

FIGURE 1
The algorithm of automatic design of SN P systems

(h) $BestFitness$ represents the best fitness through generations.

(i) $ExpectedSet$ is the expected set.

Step 2: The population of SN P systems and the fitness value are generated and caculated by the initial parameters and rules that randomly create. $F(SNPS_i)$ and $F(SNPS)$ represent the fitness value of the ith SN P system and the fitness set of all SN P systems in the population, respectively. Investigate whether SN P systems are correct according to the fitness function value of each SN P system in the population.

Step 3: The genetic algorithm is used to automatically design each SN P system in the population. $Elitism$ represents the number of reserving optimum number of SN P systems in the population. $Parent1$ and $Parent2$ are two randomly selected SN P system with high fitness value. $Crossover()$ and $Mutate()$ represent the crossover and mutate function, respectively.

Step 4: Output a new SN P system with high sensitivity and precision after completion of automatic design.

We can infer that from Figure 1, the most important three steps include building a population of SN P systems, designing a fitness function and setting elitism, crossover and mutation. We will introduce them one by one in the following subsections.

3.2 Building a population of SN P systems

A SN P system includes the number of neurons, the synapse connections between neurons, the number of rules within each neuron, the regular expressions which define each rule and the number of spikes in each neuron. A SN P system represents an individual (DNA, $SNPS_i$) in the popualtion. Here, an individual is also thought of as a set, which contains above five aspects. As a result, the building of a population of SN P systems can be divided into the following steps.

Step 1: Generate a random individual, where rules are randomly generated and other elements are predefined.

Step 2: Repeat the first step until all the individuals($SNPS_i$) in the population are produced.

Step 3: Check whether each individual is correct.

Step 4: Delete and replace individuals with incorrect and low fitness values.

Step 5: Save the initial population.

With the initial population, it is necessary to have an appropriate evaluation function to guide the population to evolve to the optimal solution. Therefore, it is worth to notice that the fitness function plays an important role throughout the automatical design process. We describe the details of the fitness function as follows.

3.3 Design of fitness function

In this subsection, we discuss how to design the fitness function, which is used to calculate the sensitivity and the precision of SN P systems. There are two data sets after the establishment of the SN P systems. One is a real output set $OutputSet$. Another is given expected set $ExpectedSet$. $OutputSet$ represents generating number set of repeating execution SN P systems for a specifical task. $ExpectedSet$ is expected number set for a special task. So a fitness function is established by comparing elements in the real output

Input: $OutputSet$, $ExpectedSet$, $tp = 0$, $fp = 0$, $fn = 0$
1: Initialization settings
2: Merging elements from $OutputSet$ and $ExpectedSet$ into $OutExSet$. The length of $OutExSet$ is n
3: $i = 1$
4: **while** $(i \leq H)$ **do**
5: $i = i + 1$
6: **if** $OutExSet(i) \in OutputSet$ **then**
7: **if** $OutExSet(i) \in ExpectedSet$ **then**
8: $tp = tp + 1$
9: Turn to **Step 21**
10: **else**
11: $fp = fp + 1$
12: Turn to **Step 21**
13: **end if**
14: **else**
15: **if** $OutExSet(i) \in ExpectedSet$ **then**
16: $fn = fn + 1$
17: Turn to **Step 21**
18: **else**
19: Turn to **Step 21**
20: **end if**
21: **end if**
22: **if** $i \geqslant n$ **then**
23: Turn to **Step 26**
24: **else**
25: Turn to **Step 4**
26: **end if**
27: $Fitness = (\frac{2 \times tp}{2 \times tp + fp + fn}) \times sf$
28: **end while**
Output: Return $Fitness$

FIGURE 2
The design of the fitness function

set and the expected set. The pseudocode of the fitness funtion is shown in Figure 2.

The category of an element in the above two sets is as follows:

(1) The output set is compared with the expected set and for every number that is in both of the sets, the true positive count tp increases.

(2) The output set is compared with the expected set and for every number that is in the output set but not in the target set, the false positive count fp increases.

(3) The output set is compared with the expected set and for every number that is not in the output set but is in the target set, the false negative count fn increases.

(4) The true negative values, those that are not in the output set and not in the target set, are not counted, as they are not needed for this design.

3.4 Elitism, crossover and mutations

DNA consists of genes, which in the case of this paper are represented by a SNP system. Each instance of DNA also contains the fitness for the genes

contained within it. The crossover function allows the exchange of genes between two parents, creating a new child DNA with the characteristics of the parents that were used. After the crossover, there is also a chance for the new child DNA to mutate, changing one of the rules in the generated network at random. To ensure variety in each population, population total new random members are added to the population pool with each generation.

Like crossover and chance for mutation, this algorithm also allows the use of elitism. This feature allows a selected number of best networks to be introduced with a new generation. This allows the algorithm to ensure that fitness will continue to increase even if poor mutations occur too frequently.

The detailed procedure of elitism, crossover and mutation are described as follows:

Elitism: Elitism, the best optimal individual in the current population, is set to 1 in the method of the automatic design, *i.e.*, a SNP system with the high sensitivity and precision can be saved to new popualtion each generation.

Crossover: The crossover is mainly composed of two steps, one is to choose the parent individuals (Parents with a higher fitness will have a higher chance of reproducing, however all parents should have a chance), and the other is to exchange the corresponding rules in the two parent individuals.

Mutations: After getting new sub-individuals from the crossover of two parent individuals, new sub-individuals are mutated and added to new population, where $MutationRate$ is adjusted according to the detailed problem dynamically. The pseudocode of dynamic adjustment is described in Figure 3.

Input: $GlobleBestFitness = 0$, $CurrentBestFitness$, $RateChange = 0$, $MutationRate = 0$
1: $i = 1$
2: **while** $(i \leq H)$ **do**
3: $i = i + 1$
4: **if** $GlobleBestFitness \leq CurrentBestFitness$ **then**
5: $GlobleBestFitness = CurrentBestFitness$
6: $RateChange++$
7: **else**
8: $RateChange = 0$
9: **end if**
10: **if** $RateChange \geq 10$ **then**
11: $MutationRate = random(0, 10)$
12: **else**
13: $MutationRate = random(10, 20)$
14: **end if**
15: **end while**
Output: Return $MutationRate$

FIGURE 3
The dynamic adjustment of mutation probability

4 EXPERIMENTATIONS AND ANALYSIS OF RESULTS

In this section, the method of the automatic design is first thoroughly tested to ensure the systems are generated as expected. First of all, a SN P system generating all even natural numbers and a SN P system generating all natural numbers are used to simulate the evolution of SN P systems towards a target configuration to ensure the sentivity and precision of the data being outputed. Secondly, we analyze the experimental results of two known SN P systems and obtain experimental conclusions.

4.1 A SN P system generating all natural numbers

A SN P system generating all natural numbers mainly contains four elements: four neurons, ten synapse connections between neurons, eight rules and two starting spikes each neuron. Out of four neurons, three neurons are general neurons and remaining one is an output neuron.

The specific sketch of a SN P system generating all natural numbers is shown in Figure 4.

In order to illustrate the performance of the method of the automatic design when simulating a SN P system generating all natural numbers, we do a dynamic behavior analysis from the fitness function value of the experimental testing process.

F_{av}: the average fitness value across ten runs. A larger value of F_{av} represents a smaller difference between the expected set and the output set.

$$F_{av} = \sum_{j=1}^{10} \sum_{i=1}^{n} F(SNPS_i) \tag{4}$$

where, $F(SNPS_i)$ represents the fitness value of ith SN P systems, n is the number of the SN P systems in the population, j represents the jth run.

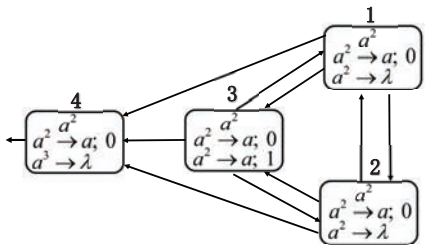

FIGURE 4
A SN P system generating all natural numbers

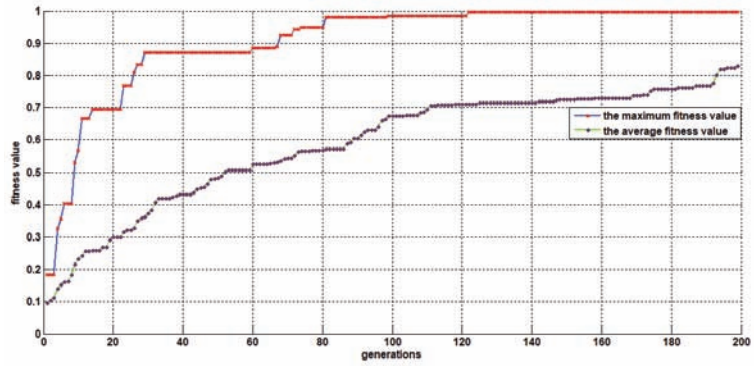

FIGURE 5
The change curves of the average fitness values and the maximum fitness values

In the process of simulated evolution, the basic design parameters are set as follows. Expected output set for the natural numbers system: 1, 2, 3, 4, 5, 6, 7, 8, 9; Population count: 4 members; Maximum number of steps per system: 50; Maximum number of repetitions per system: 50; Maximum number of generations: 200.

We obtain the change curves of the average fitness value of static and dynamic mutation probabilities in Figure 5, respectively.

As seen in Figure 5, as the number of iterations increases, the average fitness value increases and the dynamic adjustment of mutation probability is better than the static mutation probability. But when they increase to a certain value, the average fitness increases slowly.

As can be seen from Figure 6, the results of the correct natural data output are produced by a real natural SNP system and is the same as the expected set.

4.2 A SN P system generating all even natural numbers

A SN P system generating all even natural numbers mainly contains four elements: seven neurons, twelve synapse connections between neurons, twelve rules, two starting spikes in six neurons and no starting spikes in other neurons. Seven neurons divide six general neurons and one output neuron. The specific sketch of a SN P system generating all even natural numbers is shown in Figure 7.

The basic parameters are set as follows.The constraints for this set of tests, unless otherwise stated, are as follows: Expected output set for the even numbers system: 2, 4, 6, 8, 10, 12, 14, 16; Population count: 4 members, increasing by 1 every generation; Maximum number of steps per system: 50;

```
Final output set:
1       1       2       2       2       2       2       2       2       2
2       2       2       2       2       2       2       2       2       2
2       2       2       2       2       2       2       2       2       2
2       2       2       2       2       2       2       2       2       2
2       2       2       2       2       2       2       2       2       2
2       2       2       2       2       2       2       2       2       2
2       2       2       2       2       2       2       2       2       2
2       2       2       2       2       2       2       2       2       2
2       2       2       2       2       2       2       2       2       2
3       3       3       3       3       3       3       3       3       3
3       3       3       3       3       3       3       3       3       3
3       3       3       3       3       3       3       3       3       3
3       3       3       3       3       3       3       3       3       3
3       3       3       3       3       4       4       4       4       4
4       4       4       4       4       4       4       4       4       4
4       4       4       4       4       4       4       4       4       5
5       5       5       5       5       5       5       5       5       5
5       5       5       5       5       5       5       5       5       5
5       6       6       6       6       6       7       7       7       7
7       7       7       7       7       8       8       10
Press any key to exit, time elapsed: 00:00:01.5532086s
```

FIGURE 6
The output set of SN P systems generating all natural numbers

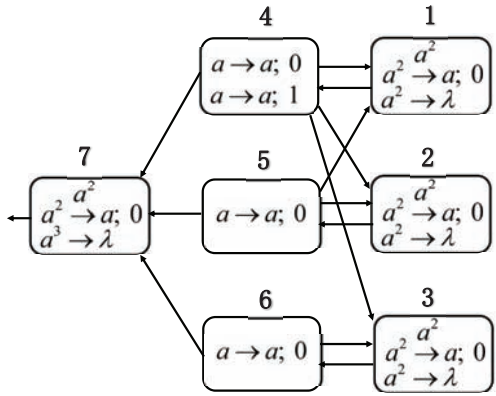

FIGURE 7
A SN P system generating all even natural numbers

Maximum number of repetitions per system: 50; Maximum number of generations: 200.

The results in Figure 8 and Figure 9 are the same as those in Figure 5 and Figure 6. The greater the number of iterations, the greater the fitness value. The output results are very close to the expected results. As a result, the expecrimental results of SN P systems generating all natural numbers and SN P systems generating all even natural numbers show that our method is feasible and effective.

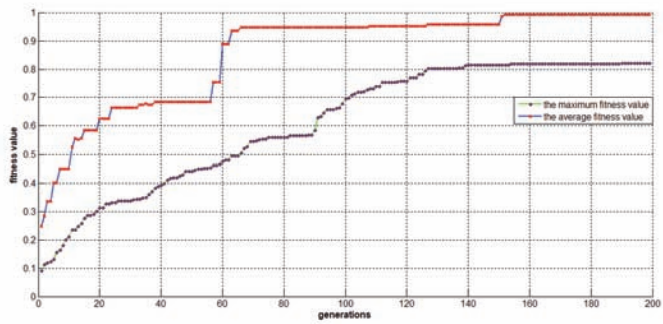

FIGURE 8
The change curves of the average fitness values and the maximum fitness values

```
Final output set:
2       2       4       4       4       4       4       4       4       4
4       4       4       4       4       4       4       4       4       4
4       4       4       4       4       4       4       4       4       4
4       4       4       4       4       4       4       4       4       4
4       4       4       4       4       4       4       4       4       4
4       4       4       4       4       4       4       4       4       4
4       4       4       4       4       4       4       4       4       4
4       4       4       4       4       4       4       4       4       4
4       4       4       4       4       4       4       4       4       4
4       4       4       4       4       4       4       4       4       4
4       4       6       6       6       6       6       6       6       6
6       6       6       6       6       6       6       6       6       6
6       6       6       6       6       6       6       6       6       6
6       6       6       6       6       6       6       6       6       6
6       6       6       6       6       6       6       6       6       6
6       6       6       6       6       6       6       8       8       8
8       8       8       8       8       8       8       8       8       8
8       8       8       8       8       8       8       10      10      10
10      10      10      10      10      10      10      12      12      12
12      12      12      12      14      14      16      24
Press any key to exit, time elapsed: 00:00:02.0664181s
```

FIGURE 9
The output set of SN P systems generating all even natural numbers

5 CONCLUSIONS

This paper proposes a clear possibility on how to use genetic algorithms to design an expected SN P system, including the overview approach design, the fitness function and the design of elitism, crossover and mutation. After randomly generating a population of SN P systems, the genetic algorithm is used to evolve the initial population to obtain the optimal SN P system. The introduction of genetic algorithm makes it easier to generate new SN P systems than manual design. Moreover, a fitness function, the comparison between the output set and the expected set, is established to reflect the precision and sensitivity of SN P systems. The experimental results show that the fitness

function designed by this study can effectively guide the search for the optimal SN P system. Finally, the experimental results from two examples show that the approach is effective to automatically design a SN P system based on genetic algorithms. In further works, we will generalize this method to generate much more SN P systems, such as generating asynchronous and time-free systems (which are usually quite hard to design manually, but relevant from an application point of view). Moreover, we have only changed the most important of these rules in presented approach, while other elements such as the number of neurons and the synapse connections between neurons do not changed in the process of evolution. Therefore, much attention will be devoted to them in furture.

ACKNOWLEDGMENT

This work was supported by the National Natural Science Foundation of China (61972324, 61672437, 61702428), the Sichuan Science and Technology Program (2018GZ0185, 2018GZ0086), New Generation Artificial Intelligence Science and Technology Major Project of Sichuan Province (2018GZDZX0044) and Artificial Intelligence Key Laboratory of Sichuan Province (2019RYJ06).

REFERENCES

[1] M. al Rifaie, M. John, and C. Suzanne. (2012). Creativity and autonomy in swarm intelligence systems. *Cognitive Computation*, 4(3):320–331.

[2] B. Aman and G. Ciobanu. (2019). Synchronization of rules in membrane computing. *Journal of Membrane Computing*, 1(4):233–240.

[3] F. Bernardini and M. Gheorghe. (2004). Population P systems. *Journal of Universal Computer Science*, 10(5):509–539.

[4] C. Buiu and A. George. (2019). Membrane computing models and robot controller design, current results and challenges. *Journal of Membrane Computing*, 1(4):262–269.

[5] M. Cavaliere and D. Genova. (2004). P systems with symport/antiport of rules. *Journal of Universal Computer Science*, 10(5):540–558.

[6] Y. Chen, G. Zhang, T. Wang, and X. Huang. (2014). Automatic design of P systems for five basic arithmetic operations within one framework. *Chinese Journal of Electronics*, 23(2):302–304.

[7] J. Cooper and R. Nicolescu. (2019). Alternative representations of P systems solutions to the graph colouring problem. *Journal of Membrane Computing*, 1(2):112–126.

[8] D. Díaz-Pernil, M. Gutiérrez-Naranjo, and H. Peng. (2019). Membrane computing and image processing: a short survey. *Journal of Membrane Computing*, 1(1):58–73.

[9] G. Escuela and M. Gutiérrez-Naranjo. (2010). An application of genetic algorithms to membrane computing. *Proceedings of the 10th Brainstorming Week on Membrane Computing*, 101–108.

[10] J. Farmer, H. Norman, and S. Alan. (1986). The immune system, adaptation, and machine learning. *Physica D*, 22(1-3):187–204.

[11] Z. Gazdag and G. Kolonits. (2019). A new method to simulate restricted variants of polarizationless P systems with active membranes. *Journal of Membrane Computing*, 1(4):251–261.

[12] M. Ionescu and Gh. Păun. (2006). Spiking neural P systems. *Fundamenta Informaticae*, 71(2):279–308.

[13] Y. Jiang, Y. Su, and F. Luo. (2019). An improved universal spiking neural P system with generalized use of rules. *Journal of Membrane Computing*, 1(4):270–278.

[14] S. Kazarlis, A. Bakirtzis, and V. Petridis. (1996). A genetic algorithm solution to the unit commitment problem. *IEEE Transactions on Power Systems*, 11(1):83–92.

[15] K. Lila and G. Rozenberg. (2008). The many facets of natural computing. *Communications of the ACM*, 51(10):72–83.

[16] C. Martín-Vide, Gh. Păun, and T. Pazos. (2003). Tissue P systems. *Theoretical Computer Science*, 296(2):295–326.

[17] R. Mayne, N. Phillips, and A. Adamatzky. (2019). Towards experimental P systems using multivesicular liposomes. *Journal of Membrane Computing*, 1(1):20–28.

[18] A. Nash and S. Kalvala. (2019). A P system model of swarming and aggregation in a myxobacterial colony. *Journal of Membrane Computing*, 1(2):103–111.

[19] D. Orellana-Martín, L. Valencia-Cabrera, A. Riscos-Núñez, and M. Pérez-Jiménez. (2019). P systems with proteins: a new frontier when membrane division disappears. *Journal of Membrane Computing*, 1(1):29–39.

[20] Z. Ou, G. Zhang, T. Wang, and X. Huang. (2013). Automatic design of cell-like P systems through tuning membrane structures, initial objects and evolution rules. *International Journal of Unconventional Computing*, 9(5):425–443.

[21] L. Pan, Gh. Păun, and G. Zhang. (2019). Foreword: Starting JMC. *Journal of Membrane Computing*, 1(1):1–2.

[22] A. Păun and B. Popa. (2006). P systems with proteins on membranes and membrane division. *Developments in Language Theory, Lecture Notes in Computer Science, Springer*, 4036:292–303.

[23] Gh. Păun. (2003). Membrane computing. *International Symposium on Fundamentals of Computation Theory*, 2751(1-3):284–295.

[24] Gh. Păun. (2006). Introduction to membrane computing. *Applications of Membrane Computing*, 2751(1-3):1–42.

[25] Gh. Păun and G. Rozenberg. (2002). A guide to membrane computing. *Theoretical Computer Science*, 287:73–100.

[26] R. Poli, J. Kennedy, and T. Blackwell. (2007). Particle swarm optimization. *IEEE Swarm Intelligence Symposium*, 1(1):33–57.

[27] H. Rong, K. Yi, G. Zhang, J. Dong, P. Paul, and Z. Huang. (2019). Automatic implementation of fuzzy reasoning spiking neural P systems for diagnosing faults in complex power systems. *Complexity*, 2019:0–16.

[28] P. Sosík. (2019). A new method to simulate restricted variants of polarizationless P systems with active membranes. *Journal of Membrane Computing*, 1(3):198–208.

[29] Y. Suzuki and H. Tanaka. (2000). chemicial evolution among arificial proto-cells. *International Conference on Artificial Life 2000, USA*, 54–63.

[30] C. Tudose, R. Lefticaru, and F. Ipate. (2012). Using genetic algorithms and model checking for P systems automatic design. *Studies in Computational Intelligence*, 344:285–302.

[31] A. Turlea, M. Gheorghe, F. Ipate, and S. Konur. (2019). Search-based testing in membrane computing. *Journal of Membrane Computing*, 1(4):241–250.

[32] T. Wang, G. Zhang, J. Zhao, Z. He, J. Wang, and M. Pérez-Jiménez. Fault diagnosis of electric power systems based on fuzzy reasoning spiking neural P systems. *IEEE Transactions on Power Systems*, 30(3):1182–1194.

[33] X. Wang, G. Zhang, F. Neri, T. Jiang, J. Zhao, and M. Gheorghe. (2016). Design and implementation of membrane controllers for trajectory tracking of nonholonomic wheeled mobile robots. *Integrated Computer-Aided Engineering*, 23(1):15–30.

[34] S. Wolfram. (1983). Statistical mechanics of cellular automata. *Review of Modern Physics*, 55(3):601–644.

[35] W. Yuan, G. Zhang, M. Pérez-Jiménez, T. Wang, and X. Huang. (2016). P systems based computing polynomials: Design and formal verification. *Natural Computing*, 15(4):591–596.

[36] G. Zhang. (2011). Quantum-inspired evolutionary algorithms: a survey and empirical study. *Journal of Heuristics*, 17(3):303–351.

[37] G. Zhang. (2012). A membrane algorithm with quantum-inspired subalgorithms and its application to image processing. *Natural Computing*, 11(4):701–717.

[38] M. Gheorghe, Gh. Păun, M. Pérez-Jiménez, and G. Rozenberg. (2013). Research frontiers of membrane computing: Open problems and research topics. *International Journal of Foundations of Computer Science*, 24(5):547–623.

[39] G. Zhang, J. Cheng, M. Gheorghe, and Q. Meng. (2013). A hybrid approach based on differential evolution and tissue membrane systems for solving constrained manufacturing parameter optimization problems. *Applied Soft Computing*, 13(3):1528–1542.

[40] G. Zhang, J. Cheng, T. Wang, and J. Zhu. (2015). Membrane computing: Theory and applications. *Beijing, China: Science Press*.

[41] G. Zhang, M. Gheorghe, L. Pan, and M. Pérez-Jiménez. (2014). Evolutionary membrane computing: a comprehensive survey and new results. *Information Sciences*, 279:528–551.

[42] G. Zhang, N. Li, and W. Jin. (2006). Novel quantum genetic algorithm and its applications. *Frontiers of Electrical and Electronic Engineering in China*, 1(1):31–36.

[43] G. Zhang and L. Pan. (2010). A survey of membrane computing as a new branch of natural computing. *Chinese Journal of Computers*, 2(30):208–214.

[44] G. Zhang, M. Pérez-Jiménez, and M. Gheorghe. (2017). Real-life applications with membrane computing. *Springer*.

[45] G. Zhang, H. Rong, F. Neri, and M. Pérez-Jiménez. (2014). An optimization spiking neural P system for approximately solving combinatorial optimization problems. *International Journal of Neural Systems*, 24(5):1–16.

[46] G. Zhang, H. Rong, Z. Ou, M. Pérez-Jiménez, and M. Gheorghe. (2014). Automatic design of deterministic and non-halting membrane systems by tuning syntactical ingredients. *IEEE Transactions on Nanobioscience*, 13(3):363–371.

[47] G. Zhang, M. Zhang, L. Pan, and M. Pérez-Jiménez. (2018). Evolutionary membrane computing: A comprehensive survey and new results. *Information Sciences*, 279:528–551.

[48] M. Zhu, G. Zhang, Q. Yang, H. Rong, W. Yuan, and M. Pérez-Jiménez. (2018). P systems-based computing polynomials with integer coefficients: design and formal verification. *IEEE Transactions on nanobioscience*, 17(3):272–280.