A Hybrid "Fast-Slow" Convergent Framework for Genetic Algorithm Inspired by Membrane Computing

Zhongwei Li[1], Shengyu Xia[1], Yun Jiang[3], Beibei Sun[1], Yuezhen Xin[1], and Xun Wang[2](✉)

[1] College of Computer and Communication Engineering, China University of Petroleum, Qingdao 266580, Shandong, China
[2] The Center for Bioengineering and Biotechnology, College of Chemical Engineering, China University of Petroleum, Qingdao 266580, Shandong, China
wangsyun@upc.edu.cn
[3] School of Computer Science and Information Engineering, Chongqing Technology and Business University, Chongqing 400067, China

Abstract. Genetic algorithm is a well known bio-inspired algorithm, which has been widely used to solve practical problems in real-life. The performance of the algorithm heavily depends on the convergence related to the values of parameters involved. It is formulated as a hard problem to select suitable values of mutation and crossover rates to achieve fast or slow convergence for unknown problems. As a new study of system framework inspired by cell model, membrane computing models is with a membrane structure having region segmentation, intrinsic discrete, non-deterministic, programmable and transparent features. In this paper, a hybrid "fast-slow" convergent framework for genetic algorithm inspired by membrane computing is proposed and applied to search optimal solution of 41 benchmark functions. It is obtained by the data experimental results that our method performs well in solving benchmark functions by achieving accuracy rate about 96%.

Keywords: Membrane computing · Genetic algorithm · Membrane structure · Convergence

1 Introduction

Genetic algorithm (GA) is bio-inspired intelligent algorithm abstracted from the human evolving process. Nowadays, the algorithm is known adaptive, heuristic, iterative, and has been applied in solving plenty of practical problems. The performance of GA heavily depends on the values of involved parameters, such as mutation rate and crossover rate of the population. The most intuitive case is that with different values, the convergence which is an important indicator of performance to test the algorithm, will be quite different. High convergence rate does not mean it can search for the best solutions; while sometimes, slower convergence can get better results, but it means spending more time [1].

Due to the instability of the convergence, many researchers are focus on the precocious convergence of GA. It is proposed in [2] the genetic markers to actively avoid convergence to a particular rooted tree structure. This is achieved by maintaining a number of unique genetic markers in the population. After that, the structure fitness sharing (SFS) algorithm proposed in [3,4] is taken as a possible way to attempt to promote diversity based on tree structure. Motivated by the fitness sharing concept, it uses labels on tree structures to decrease the fitness of structures that are over-represented in the population. Generally speaking, most of the researches solve this problem by optimizing algorithm and intermediate data processing.

Membrane computing, initialed by Gh Paun in 1998 is known as new branch of natural computing [5]. The systems investigated in the framework of membrane computing is called P systems, and plenty of P systems have been developed, including cell-like P systems, tissue P systems and spiking neural P systems [6–20]. In this work, we propose a new model inspired from membrane computing models to achieve "fast-slow" convergence rate of GA in the membrane structure. The obtained algorithm is a new candidate in membrane algorithm, and many researchers have done good works on it. Currently, membrane computing has been used in optimization field [31], Systems and Synthetic Biology [21], Troubleshooting [36], economics [37] and linguistics [38]. These experiments demonstrate that applying Membrane Algorithm to optimize Genetic Algorithm is feasible. We developed here a thread control process following the Nested Membrane System [24] to searching optimal solution, where a single GA is used in each membrane and performs as a thread in the program [22]. After one iteration (evolution), the population will produce the best individual. Under the control of the communication rule [23,36], the efficiency of searching optimal solution gets a big promotion, when the problem has no solving information. It is obtained by the data experimental results that our method performs well in solving 41 benchmark functions by achieving accuracy rate about 96%.

2 Related Technologies

2.1 Genetic Algorithm

Genetic Algorithm (GA) was first proposed by J. Holland in 1975 [26]. It is a type of heuristic random search method inspired by natural selection and genetic mechanism of biological evolution law (survival of the fittest). It contains feature is the direct operating to the structured objects without the delimitation of derivation and continuity of function. It is inherent implicit parallelism and better global optimization and it can automatically obtain and guide optimized searching space for adjusting the search direction.

In general computing process, GA is started with setting the potential solution (population), and a population is consists of genes encoded by a certain number of individuals (individual). It is needed initially to encode individuals for simplify computer operation, such as binary encoding. After producing the first generation of populations, each generation produce more good approximate solution in accordance with the principle of survival of the fittest. In each generation, select individual according to the individual’s fitness size, and then generate a population representative of the new solution set by genetic operators combined with cross and mutation. This process will lead new population to be more adapted to the environment, and the last population of the best individual can be the approximate optimal solution after decoding [25].

2.2 Membrane Computing Inspired Algorithm

Membrane Computing is a new branch of natural computing. It is originated from natural cells, and the structure also builds on the biological cells. The systems investigated in membrane computing are named P systems, which is defined as a series of membrane structures containing chemical substances (limited number), catalyst and rules (including the rules of the reaction, membrane transport rules etc.). It is shown in Fig. 1 the membrane structure of the P systems. Like in real biological cells, when the reactants (sometimes catalyst) are contact with each other, the chemical reaction will occur. Due to the random applications of rules, the calculation will be uncertain, in the other words, the repetition of the same question may lead to multiple solutions. When the computation of the P system is completed, the chemicals exist out of the outermost membrane will reach steady state, which means no reaction will continue.

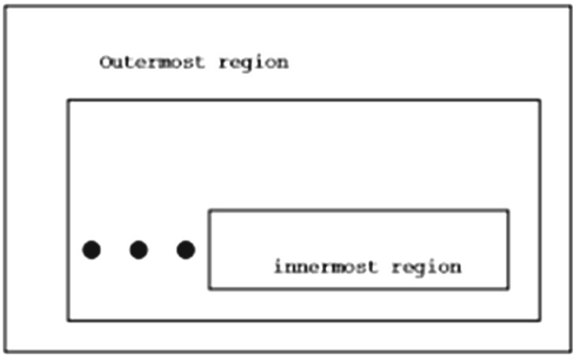


Fig. 1. Membrane structure.

A membrane algorithm framework consists of three different kinds of components:

- A number of regions which are separated by nested membranes (Fig. 1).
- For every region, a subalgorithm and a few tentative solutions of the optimization problem to be solved.
- Solution transporting mechanisms between adjacent regions.

There are three basic types of membrane system: Cell-like P system, Tissue P system, spiking neural P systems [29,32,35]. We consider here a cell-like membrane structure named Nested Membrane System, which is a friendly model for programming. The structure is used by Nishida to solve the TSP problem [30,31], and have been applied for data optimization [31].

We denote by S a feasible solution of the problem, which is distributed differences in different membranes. The communication rules means that the membrane sends some solutions into the outer membrane which directly contains it. The rule can be written as follows.

$$\{a_{max1}, a_{max2}, \ldots, a_{maxn}\}_i - \{\}_i a_{max1}, a_{max2}, \ldots, a_{maxn} \tag{1}$$

It is denoted by i the membrane i, and by $amax_1, amax_2, \ldots, amax_n$ the n best solutions in the region $1, 2, \ldots, n$, respectively. The model converges very fast because of the communication between membranes. In terms of realization of membrane computing, some associated simulation software have been released.

3 The Model and Data Experiments

3.1 GA Program

We design a basic GA program to run the dimensional function. In the program, we can set initial conditions to control the convergence rate. Every gene contains the potential solution and the threshold. In order to simplify the crossing and mutation process, encoding process is omitted as in [33].

- Chromosome: Chromosome and can be called individuals, a certain number of individuals of the population, the number of groups of individuals called population size.
- Gene: Gene elements include characteristics of the individual genes. In this paper, a set of possible solutions $S = (x1, x2, x3)$ are designed and each of them is called gene.
- Fitness: Each individual's degree of adaptation to the environment is called fitness. In order to reflect the ability to adapt to the chromosome, the introduction of the function of each chromosome in question can be measured. Here, the function is calculated to value in the population of individuals.
- Select: Select means winning individuals from population, and Selecting operation is based on the population of individual fitness assessment.
- Cross: Genetic recombinant (plus variation) play a central role in the process of evolution is a genetic recombinant organisms (plus variation). Because of giving up Genes encoding process gene cross is implemented by exchanging a random parameter of two genes.
- Variation: The basic contents of the gene mutation operator are the value of a population of some individual strings locus for change. Here we use Real value variation.

3.2 Membrane Structure

The nested membrane structure of degree m is selected, which means the number of membrane is m. The value of m is set to be 5 or 6 here, and in each membrane, GA is performed with different mutation and crossover rates. All of the GA run the same function. Because of the lightweight program, we set each GA as a thread [34]. The data communication rule works in neighbor membrane. The communication process looks as follows (current membrane is the middle membrane):

1. Each membrane start GA thread
2. Suspend thread every 50 iteration
3. Monitoring inner membrane
   If (inner membrane has no request for communication):
   - 3.1 Compared with the individual of the outer membrane
   - 3.2 Suspend outer membrane
   - 3.3 Replace the best individual of outer membrane
   - 3.4 Reuse thread
   - 3.5 Judge the outer membrane to reuse (avoid the thread to be suspended before reaching 50 iteration by inner membrane)
     Else: wait for the inner membrane
4. The best individual in the outmost becomes the output of the algorithm

The individual are modified only when both of the membranes have been suspended. In order to avoid deadlocks, the inner membrane has higher priority than the outer membrane.

3.3 Data Experiments

It is tested the proposed method by solving 41 benchmark functions. In the step of initialization, we created 5 GA and set different initialization information. The iteration is set to be 10000 for each GA. We controlled convergence rates by changing number of individuals, cross rate and variation rate (Table 1).

Table 1. The values of involved parameters in the GA in different regions

Population	Iterations	Cross	Variation
1000	10000	0.8	0.08
1000	10000	0.6	0.01
500	10000	0.8	0.08
500	10000	0.6	0.01
800	10000	0.8	0.1

The interface of the software is shown in Fig. 2.

The tested functions are listed in Table 2.

It is obtained by the data experimental results that our method performs well in solving benchmark functions by achieving accuracy rate about 96%.

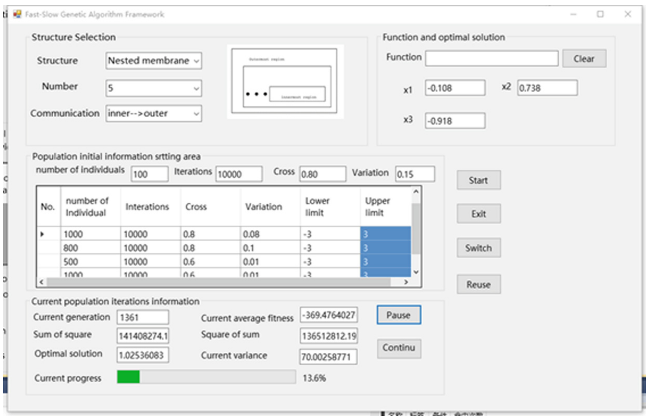
We exam the formulas the help of the model (Table 3).

Table 2. The list of tested benchmark functions

Formula	
Ackley's function (Ak)	$f(x,y) = -20\exp(-0.2\sqrt{0.5(x^2+y^2)}) - \exp(0.5(\cos(2(\pi)x) + \cos(2(\pi)y))) + e + 20$
Sphere function (Sh)	$f(x) = \sum_{i=1}^{n} x_i^2$
Rosenbrock function (Rbk)	$f(x) = \sum_{i=1}^{n-1}[100(x_{i+1} - x_i^2) + (x_i - 1)^2]$
Beale's function (Bl)	$f(x,y) = (1.5-z+xy)^2+(2.25-x+xy^2)^2+(2.625-x+xy^3)^2$
GoldsteinPrice function (GP)	$f(x,y) = (1 + (x+y+1)^2(19 - 14x + 3x^2 - 14y + 6xy + 3y^2)) \times (30+(2x-3y)^2(18-32x+12x^2+48y-36xy+27y^2))$
Booth's function (Bt)	$f(x,y) = (x+2y-7)^2 + (2x+y-5)^2$
Bukin function N.6 (BN.6)	$f(x,y) = 100\sqrt{\lvert y - 0.01x^2\rvert} + 0.01\lvert x+10\rvert$
Matyas function (Mt)	$f(x,y) = 0.26(x^2+y^2) - 0.48xy$
Levi function N.13 (LN.13)	$f(x,y) = \sin^2(3\pi x) + (x-1)^2(1+\sin^2(3\pi y)) + (y-1)^2(1+ sin^2(2\pi y))$
Three-hump camel function(Thc)	$f(x,y) = 2x^2 - 1.05x^4 + \frac{x^6}{6} + xy + y^2$
Easom function (Es)	$f(x,y) = -\cos x \cos y \exp(-((x-\pi)^2 + (y-\pi)^2))$
Cross-in-tray function (Ct)	$f(x,y) = -0.0001(\lvert \sin x \sin y \exp(\lvert 100 - \frac{\sqrt{x^2+y^2}}{\pi}\rvert)\rvert + 1)^{0.1}$
Eggholder function (Ehd)	$f(x,y) = -(y+47)\sin(\sqrt{\lvert \frac{x}{2} + (y+47)\rvert}) - x\sin(\sqrt{\lvert x - (y+47)\rvert})$
Holder table function (Ht)	$f(x,y) = -\lvert \sin x \cos y \exp(\lvert 1 - \frac{\sqrt{x^2+y^2}}{\pi}\rvert)\rvert$
McCormick function (McC)	$f(x,y) = \sin(x+y) + (x-y)^2 - 1.5x + 2.5y + 1$
Schaffer function N. 2 (Scf.2)	$f(x,y) = 0.5 + \frac{\sin^2(\lvert x^2-y^2\rvert)-0.5}{(1+0.001(x^2+y^2))^2}$
Schaffer function N. 4 (Scf.4)	$f(x,y) = 0.5 + \frac{\cos^2(sin(\lvert x^2-y^2\rvert))-0.5}{(1+0.001(x^2+y^2))^2}$
Hump Functions (Hmp)	$f(x) = 4x_1^2 - 2.1x_1^4 + x_1^6/3 + x_1x_2 - 4x_2^2 + 4x_2^4$
Rastrigin function (Rst)	$f(x) = An + \sum_{i=1}^{n}(x_i^2 - A\cos(2\pi x_i))$
Colville function (Clv)	$f(x) = 100(x_1^2 - x_2)^2 + (x_1-1)^2 + (x_3-1)^2 + 90(x_3^2 - x_4)^2 + 10.1((x_2-1)^2 + (x_4-1)^2) + 19.8(x_2^- 1)(x_4-1)$
Griewank function (Gwk)	$f(x) = \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n}\cos(\frac{x_i}{\sqrt{i}}) + 1$
Schwefel function (Swf)	$f(x) = -\sum_{i=1}^{n} x_i \sin(\sqrt{\lvert x_i\rvert})$
Shubert function (Shb)	$f(x) = (\sum_{i=1}^{5} i\cos((i+1)x_1+i))(\sum_{i=1}^{5} i\cos((i+1)x_2+i))$
Sum Squares function (SSq)	$f(x) = \sum_{i=1}^{n} ix_i^2$
Zakharov function (Zkr)	$f(x) = \sum_{i=1}^{n} x_i^2 + (0.5\sum_{i=1}^{n} ix_i)^2 + (0.5\sum_{i=1}^{n} ix_i)^4$
Generalized Rastrigins function (GR)	$f(x) = \sum_{i=1}^{n}[x_i^2 - 10\cos(x\pi x_i) + 10]$
Styblinski-Tang function (SbT)	$f(x) = 0.5(\sum_{i=1}^{n} x_i^4 - 16x_i^2 + 5x_i)$
Michaelwiczs function (Mcw)	$f(x,y) = -\sin x \sin^{20}(\frac{x^2}{\pi}) - \sin y \sin^{20}(\frac{2y^2}{\pi})$
Six-hump camel back function (Shcb)	$f(x,y) = (4 - 2.1x^2 + \frac{1}{3}x^4)x^2 + xy + 4(y^2-1)y^2$
Xin-She Yangs functions (XSY)	$f(x) = (\sum_{i=1}^{n} \lvert x_i\rvert)\exp(-\sum_{i=1}^{n}\sin(x_i^2))$
J.D. Schaffer function (JDS)	$f(x) = \frac{\sin^2(\sqrt{(x_1^2+x_2^2)})-0.5}{[1+0.001(x_1^2+x_2^2)]^2} - 0.5$
Quartic Function i.e. Niose (Qie)	$f(x) = \sum_{i=1}^{n} ix_i^4 + random[0,1)$
Step function (Step)	$f(x) = \sum_{i=1}^{n}(\lvert x_i + 0.5\rvert)^2$
Schwefels Problem 2.21 (Swf2.21)	$f(x) = \max_{i=1}^{n}\{\lvert x_i\rvert\}$
Schwefels Problem 2.22 (Swf2.22)	$f(x) = \sum_{i=1}^{n}\lvert x_i\rvert + \prod_{i=1}^{n}\lvert x_i\rvert$
Schwefels Problem 1.2 (Swf1.2)	$f(x) = \sum_{i=1}^{n}(\sum_{j=1}^{n} x_j)^2$

Table 3. The values of parameters in tested benchmark functions

Formula	Search domain	Minimum	Result	Deviation
Ak	[–10,10] n = 2	0	0	0
Sh	[–10,10] n = 2	0	0	0
	[–10,10] n = 30	0	0	0
Rbk	[–10,10] n = 2	0	0	0
	[–512,512] n = 10	0	0	0
Bl	[–10,10]	0	0	0
GP	[–2,2]	3	0	Err
Bt	[–10,10]	0	0	0
BN.6	[–15,15]	0	0	0
Bt	[–10,10]	0	0	0
Mt	[–10,10]	0	0	0
Es	[–100,100]	–1	–0.98564252	1.5%
LN.13	[–10,10]	0	0	0
The	[–10,10]	0	0	0
Ct	[–10,10]	–2.06261	–2.07697545	0.69%
Ehd	[–512,512]	–959.6407	–954.20825867	0.566%
Ht	[–10,10]	–19.2085	–19.12593522	0.43%
McC	[–3,4]	–1.9133	–1.89645318	0.88%
Scf. 2	[–5,5]	0	0	0
Scf. 4	[–5,5]	0.292579	0	Err
Hmp	[–5,5] n = 2	0	0	0
Rst	[–5.12,5.12] n = 2, A = 10	0	0	0
	[–5.12,5.12] n = 10, A = 10	0	0	0
Clv	[–10,10] n = 4	0	0	0
	[–10,10] n = 10	0	0	0
Gwk	[–600,600] n = 2	0	0	0
	[–600,600] n = 10	0	0	0
Swf	[–500,500] n = 2	837.9658	837.96552803	$3.24e^{-7}$
	[–500,500] n = 10	0	0	0
Shb	[–10,10] n = 3	–186.7309	–186.72187594	0.0048%
SSq	[–10,10] n = 2	0	0	0
	[–10,10] n = 30	0	0	0
Zkr	[–5,10] n = 2	0	0	0
	[–5,10] n = 10	0	0	0
GR	[–5.12,5.12] n = 2	0	0	0
	[–5.12,5.12] n = 10	0	0	0
SbT	[–5,5] n = 2	(–78.33234,–78.33232)	–78.322614322	0.023%
	[–5,5] n = 10	(–391.6617,–391.6616)	–391.59810786	0.0162%
Mcw	[0,5] m = 10, n = 2	−1.8013	−1.80120638	0.0052%
Shcb	[–3,3]	–1.0316	1.03052387	0.10%
XSY	[–2,2] n = 2	0	0	0
JDS	[–100,100]	–1	0.99022143	0.978%
Qie	[–1.28,1.28]	0	0	0
Step	[–100,100] n = 3	0	0	0
	[–100,100] n = 10	0	0	0
Swf2.21	[–100,100] n = 3	0	0	0
	[–100,100] n = 100	0	0	0
Swf2.22	[–10,10] n = 3	0	0	0
	[–10,10] n = 10	0	0	0
Swf1.2	[–100,100] n = 3	0	0	0


Fast-Slow Genetic Algorithm Framework

Structure Selection

Structure: Nested membrane

Number: 5

Communication: inner-->outer

Function and optimal solution

Function:

x1: -0.108 x2: 0.738

x3: -0.918

Population initial information srtting area

number of individuals: 100 Iterations: 10000 Cross: 0.80 Variation: 0.15

No.	number of Individual	Interations	Cross	Variation	Lower limit	Upper limit
▸	1000	10000	0.8	0.08	-3	3
	800	10000	0.8	0.1	-3	3
	500	10000	0.6	0.01	-3	3
	1000	10000	0.6	0.01	-3	3

Current population iterations information

Current generation: 1361 Current average fitness: -369.4764027

Sum of square: 141408274.1 Square of sum: 136512812.19

Optimal solution: 1.02536083 Current variance: 70.00258771

Current progress: 13.6%

Fig. 2. Fast-slow GA framework.

4 Conclusion

In this paper, a hybrid "fast-slow" convergent framework for genetic algorithm inspired by membrane computing is proposed. Such framework incorporates basic Cell-like P System and GA. Several basic features like compartmentalization, communication among compartments, dynamic membrane structure help GA to combine the convergence. It is tested the proposed method by solving 41 benchmark functions. It is found that our method performs well in solving benchmark functions by achieving accuracy rate about 96%.

Compared with these results, we can find the algorithm show good performance for searching optimal solution. It combines the potential results of different GA and provides a method to solve premature convergence. It also reduces the influence of the initialization to GA. On the other hand, the communication rule can be optimized, and the present paper control the data transmission by making use of thread control inspired by Nested membrane structure.

Membrane algorithms inherit the parallelism of P system. In the further study, the algorithms will be naturally implemented on a parallel hardware. The parallelism is simulated in a common serial machine. The GA in each membrane is not true parallel processing and it is also the difficulty of the application of membrane computing, even if the algorithm running in a cluster, because the communication costs is too high to optimization. So, there are still many improvements to do if the framework in this paper runs on a parallel hardware, such as GPU. We hope other membrane structure such as spiking neural P systems [39] can also be applied if the threads control method is well designed. It is of interests to replace GA in each membrane by some other intelligent algorithms, such as PSO, simulated annealing. As well, some other membrane structures, for instance, star membrane structure, and rooted membrane structure can be expanded to our hybrid framework.

Acknowledgment. This work was supported by National Natural Science Foundation of China (61402187, 61502535, 61572522, 61502063 and 61572523), China Postdoctoral Science Foundation funded project (2016M592267), Program for New Century Excellent Talents in University (NCET-13-1031), 863 Program (2015AA020925), Natural Science Foundation Project of CQ CSTC (No. cstc2012jjA40059), and Fundamental Research Funds for the Central Universities (247201607005A).

References

1. Lin, F., Zhou, C., Changm K.: Convergence rate analysis of allied genetic algorithm. In: IEEE Conference on Decision and Control, pp. 786–791 (2010)
2. Burks, A.R., Punch, W.F.: An efficient structural diversity technique for genetic programming. In: ACM Conference on Genetic and Evolutionary Computation, pp. 991–998 (2015)
3. Hu, J., Seo, K., Li, S., et al.: Structure fitness sharing (SFS) for evolutionary design by genetic programming. In: Genetic & Evolutionary Computation Conference, pp. 780–787 (2012)
4. Mckay, R.I.: Fitness sharing in genetic programming. In: Genetic and Evolutionary Computation Conference, pp. 10–12 (2000)
5. Paun, G.: Membrane Computing: An Introduction (2002)
6. Song, T., Pan, L., Wang, J., Venkat, I., Subramanian, K., Abdullah, R.: Normal forms of spiking neural P systems with anti-spikes. IEEE Trans. NanoBiosci. **11**(4), 352–359 (2012)
7. Padmavati Metta, V., Kelemenova, A.: Universality of spiking neural P systems with anti-spikes. New Math. Nat. Comput. **8**(3), 281–283 (2014)
8. Krithivasan, K., Metta, V.P., Garg, D.: On string languages generated by spiking neural P systems with anti-spikes. Int. J. Found. Comput. Sci. **22**(1), 15–21 (2011)
9. Song, T., Wang, X., Zhang, Z., Chen, Z.: Homogenous spiking neural P systems with anti-spikes. Neural Comput. Appl. doi:10.1007/s00521-013-1397-8
10. Song, T., Liu, X., Zeng, X.: Asynchronous spiking neural P systems with anti-spikes. Neural Process. Lett. **42**(3), 633–647 (2014)
11. Jiang, K., Pan, L.: Spiking neural P systems with anti-spikes working in sequential mode induced by maximum spike number. Neurocomputing **171**(1), 1674–1683 (2015)
12. Metta, V.P., Raghuraman, S., Krithivasan, K.: Spiking neural P systems with cooperating rules. In: Gheorghe, M., Rozenberg, G., Salomaa, A., Sosík, P., Zandron, C. (eds.) CMC 2014. LNCS, vol. 8961, pp. 314–329. Springer, Heidelberg (2014). doi:10.1007/978-3-319-14370-5_20
13. Metta, V.P., Raghuraman, S., Krithivasan, K.: Small universal spiking neural P systems with cooperating rules as function computing devices. In: Gheorghe, M., Rozenberg, G., Salomaa, A., Sosík, P., Zandron, C. (eds.) CMC 2014. LNCS, vol. 8961, pp. 300–313. Springer, Heidelberg (2014). doi:10.1007/978-3-319-14370-5_19
14. Song, T., Xu, J., Pan, L.: On the universality and non-universality of spiking neural P system with rules on synapses. IEEE Trans. NanoBiosci. **14**(8), 960–966 (2015)
15. Song, T., Pan, L.: Spiking neural P systems with rules on synapses working in maximum spiking strategy. IEEE Trans. Nanobiosci. **14**(4), 465–477 (2015)
16. Song, T., Pan, L.: Spiking neural P systems with rules on synapses working in maximum spikes consumption strategy. IEEE Trans. Nanobiosci. **14**(1), 38–44 (2015)
17. Song, T., Zou, Q., Liu, X., Zeng, X.: Asynchronous spiking neural P systems with rules on synapses. Neurocomputing **151**, 1439–1445 (2015)

18. Zhang, X., Zeng, X., Pan, L.: Weighted spiking neural P systems with rules on synapses. Fundamenta Informaticae **134**(1–2), 201–218 (2014)
19. Ionescu, M., Paun, G., Pérez-Jiménez, M.J., Yokomori, T.: Spiking neural dP systems. Fundamenta Informaticae **111**(4), 423–436 (2011)
20. Song, T., Pan, L.: Spiking neural P systems with request rules. Neurocomputing (2016). doi:10.1016/j.neucom.2016.02.023
21. Graham, S., Saxton, J., Woodward, M., et al.: Applications of membrane computing in systems and synthetic biology. Emergence Complex. Comput. **7**(09), S624 (2013)
22. Moon, S., Chang, B.M.: A thread monitoring system for multithreaded java programs. ACM Sigplan Not. Homepage **41**(5), 21–29 (2006)
23. Wang, X., Song, T., Gong, F., Zheng, P.: On the computational power of spiking neural P systems with self-organization. Sci. Rep. doi:10.1038/srep27624
24. Leporati, A., Pagani, D.: A membrane algorithm for the min storage problem. In: Hoogeboom, H.J., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2006. LNCS, vol. 4361, pp. 443–462. Springer, Heidelberg (2006). doi:10.1007/11963516_28
25. Pezzella, F., Morganti, G., Ciaschetti, G.: A genetic algorithm for the flexible job-shop scheduling problem. Comput. Oper. Res. **35**(10), 3202–3212 (2008)
26. Goldberg, D.E.: Genetic algorithm in search
27. Paun, G., Rozenberg, G.: A guide to membrane computing. Theor. Comput. Sci. **287**(1), 73–100 (2002)
28. Paun, G.: Computing with Membranes, Working with Computers, pp. 108–143. National Computing Centre Limited (1982)
29. Ionescu, M., Paun, G., Yokomori, T.: Spiking neural P systems with an exhaustive use of rules. Int. J. Unconventional Comput. **3**, 135–153 (2007)
30. 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). doi:10.1007/11603047_4
31. Huang, L., Wang, N.: An optimization algorithm inspired by membrane computing. In: Jiao, L., Wang, L., Gao, X., Liu, J., Wu, F. (eds.) ICNC 2006. LNCS, vol. 4222, pp. 49–52. Springer, Heidelberg (2006). doi:10.1007/11881223_7
32. Gutierrez-Naranjo, M.A., Perez-Jimenez, M.J., Ramrez-Martnez, D.: A software tool for verification of spiking neural P systems. Nat. Comput. **7**(4), 485–497 (2008)
33. Wu, Y., Tang, Y., Han, B., et al.: A topology analysis and genetic algorithm combined approach for power network intentional islanding. Int. J. Electr. Power Energy Syst. **71**, 174–183 (2015)
34. Nowotniak, R., Kucharski, J.: GPU-based tuning of quantum-inspired genetic algorithm for a combinatorial optimization problem. Bull. Pol. Acad. Sci. Tech. Sci. **60**(2), 323–330 (2012)
35. Song, T., Pan, L.: Spiking neural P systems with rules on synapses working in maximum spiking strategy. IEEE Trans. Nanobiosci. **14**(1), 465–477 (2015)
36. Wang, T., Zhang, G., Zhao, J., et al.: Fault diagnosis of electric power systems based on fuzzy reasoning spiking neural P systems. IEEE Trans. Power Syst. **30**, 1182–1194 (2014)
37. Un, G., Un, R.: Membrane computing and economics: numerical P systems. Fundamenta Informaticae **73**(1–2), 213–227 (2006)
38. Ciobanu, G., Păun, G., Prez-Jimnez, M.J.: Applications of membrane computing. Nat. Comput. **287**(1), 73–100 (2006)
39. Zhang, G., Rong, H., Neri, F.: An optimization spiking neural P system for approximately solving combinatorial optimization problems. Int. J. Neural Syst. **24**(5), 1440006 (2014)