AN OPTIMIZATION SPIKING NEURAL P SYSTEM FOR APPROXIMATELY SOLVING COMBINATORIAL OPTIMIZATION PROBLEMS

GEXIANG ZHANG* and HAINA RONG
School of Electrical Engineering
Southwest Jiaotong University
Chengdu, 610031, China

**zhgxdylan@126.com*

FERRANTE NERI

Centre for Computational Intelligence
De Montfort University, Leicester, UK
fneri@dmu.ac.uk

MARIO J. PÉREZ-JIMÉNEZ
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012, Spain

Membrane systems (also called P systems) refer to the computing models abstracted from the structure and the functioning of the living cell as well as from the cooperation of cells in tissues, organs, and other populations of cells. Spiking neural P systems (SNPS) are a class of distributed and parallel computing models that incorporate the idea of spiking neurons into P systems. To attain the solution of optimization problems, P systems are used to properly organize evolutionary operators of heuristic approaches, which are named as membrane-inspired evolutionary algorithms (MIEAs). This paper proposes a novel way to design a P system for directly obtaining the approximate solutions of combinatorial optimization problems without the aid of evolutionary operators like in the case of MIEAs. To this aim, an extended spiking neural P system (ESNPS) has been proposed by introducing the probabilistic selection of evolution rules and multi-neurons output and a family of ESNPS, called optimization spiking neural P system (OSNPS), are further designed through introducing a guider to adaptively adjust rule probabilities to approximately solve combinatorial optimization problems. Extensive experiments on knapsack problems have been reported to experimentally prove the viability and effectiveness of the proposed neural system.

Keywords: Membrane computing; spiking neural P system; extended spiking neural P system; optimiza-tion spiking neural P system; knapsack problem.

1. Introduction

Inspired by the central nervous systems of animals, artificial neural networks (ANNs) in computer science and related fields refer to a class of computational models consisting of interconnected neurons.[15,18] ANNs are capable of machine learning and pattern recognition through computing values from inputs by feeding information through the network. In the past three decades, ANNs have been widely used in various fields, such as classification,[9] earthquake prediction,[6,53,54] epilepsy and seizure detection,[29,30] and optimization,[1–5,7,8,55,56,67,73] due to their outstanding characteristics of self-adaptability, self-organization and real-time learning capability.

ANNs can be classified into three different generations in terms of their computational units.[48]

The first generation is characterized by *McCulloch–Pitts neurons*, which are also referred to perceptrons or threshold gates, as computational units. Several typical examples are multilayer perceptrons (also called threshold circuits), Hopfield nets, and Boltzmann machines. The main limitation of the first generation ANNs is that they can only output *digital* results and therefore can process only Boolean functions.[12,48,68] The computational units in the second generation ANNs use an *activation function* with a *continuous* set of possible output values to a weighted sum (or polynomial) of the inputs. This kind of neural networks support learning algorithms based on gradient descent such as backpropagation. Feedforward, recurrent sigmoidal neural nets and radial basis function neural networks are representative paradigms. The second generation ANNs are able to deal with *analog* input and output and compute arbitrary boolean functions with the help of thresholding at the network output. The main problem of the second generation is that the *firing rate* biological interpretation, i.e. the output of a sigmoidal unit as a representation of the current firing rate of a biological neuron, is questionable, see Refs. 12, 48 and 68.

The experimental evidence, accumulated during the last few years, that many biological neural systems use the timing of single action potentials (or "spikes") to encode information have lead to the third generation of neural networks, which apply spiking neurons (or "integrate-and-fire neurons") as computational units, called spiking neural networks (SNNs).[48] SNNs, which were introduced in Refs. 46 and 47 and are composed of spiking neurons communicating by sequences of spikes, use *time differences* between pulses to encode information and are able to process substantial amount of information with a relatively small number of spikes.[28,60] As both computationally powerful and biologically more plausible models of neuronal processing, SNNs are increasingly receiving renewed attention, due to the incorporation of the concept of time into their operating model in addition to neuronal and synaptic states.[12,68] Typical SNN models found in the literature are Hodgkin and Huxley (HH), FitzHugh–Nagumo (FHN), integrate-and-fire (IF), leaky integrate-and-fire (LIF), Spike Response Model (SRM), Izhikevich model (IM) and Morris–Lecar (ML), see Ref. 68. To date, SNNs have been widely investigated in various aspects, such as fundamental issues like biologically plausible models[10,39,49,50,64,66,74,75] and training algorithms,[26,72,79] hardware/software implementation,[41,65] and wide applications.[11,27,45,69]

The most attractive feature that SNNs use *time* to encode information is very useful to develop a novel type of membrane systems (also called P systems), which refer to the computing models abstracted from the structure and the functioning of the living cell as well as from the cooperation of cells in tissues, organs, and other populations of cells. This area of membrane computing was initiated by Păun[57] and listed by Thompson Institute for Scientific Information (ISI) as an emerging research front in computer science in 2003. Since then, membrane computing becomes a branch of natural computing and has developed very fast into a vigorous scientific discipline. P systems use "symbol" to encode information, except for spiking neural P systems (SNPS), which was introduced by Ionescu *et al.*[40] in 2006 and is the incorporation of the idea of spiking neurons into the area of membrane computing. SNPS can be also considered as the inspiration of the combination of SNNs and membrane computing models. Nowadays, much attention is paid to SNPS from the perspectives of theory and applications because they are the newest and promising type of membrane systems except for cell- and tissue-like P systems.

Among the various investigations on membrane computing, the attempt to extend a P system to approximately solve an optimization problem is one of the most promising and important research directions, see Refs. 58 and 87, as it would allow a further automatism of machines, see Refs. 13, 14, 37 and 38. The combination of a P system framework with meta-heuristic algorithms[42] dates back to the year of 2004, when Nishida combined a nested membrane structure with a tabu search to solve traveling salesman problems.[51] Subsequently, this kind of approaches, called membrane-inspired evolutionary algorithms (MIEAs),[84,87] has gone through a fast development. In Ref. 35, a hybrid algorithm combining P systems and genetic algorithms (GAs) was presented to solve multi-objective numerical optimization problems. In Ref. 81, an MIEA integrating a one-level membrane structure (OLMS) with

a quantum-inspired evolutionary algorithm (QIEA), called QEPS, was proposed to solve knapsack problems. This membrane structure was also combined with a QIEA and tabu search,[88] differential evolution (DE),[17] ant colony optimization,[83] particle swarm optimization (PSO)[89] and multiple QIEA components[86] to solve the time–frequency atom decomposition problem of radar emitter signals, numerical optimization problems, traveling salesman problems, broadcasting problems in P systems and image processing problems, respectively. In Refs. 76 and 77, DNA sequences design was optimized by designing an MIEA based on crossover and mutation rules and a dynamic MIEA combining the fusion and division rules of P systems with active membranes and search strategies of DE and PSO, respectively. In Refs. 36, 78 and 80, hybrid MIEAs were presented to solve constrained optimization problems, the proton exchange membrane fuel cell model parameter estimation problems and a controller design problem of a time-varying unstable plant, respectively. In Ref. 85, a tissue membrane system with a network structure was used to appropriately organize five representative DE variants for solving constrained manufacturing parameter optimization problems. These investigations clearly indicate the necessity and feasibility of the use of P systems for various engineering optimization problems. In Ref. 58, an argument that MIEAs could be used in practice to tackle real-world optimization problems has been made.

On the other hand, all the MIEAs currently present in the literature make use of hierarchical or network membrane structures of P systems to properly organize evolutionary operators of heuristic approaches in order to attain the solution of optimization problems. In other words, the current state-of-the-art considers MIEAs as hybrid methods that, when evolutionary operators are integrated within them, can be used for solving optimization problems.

A SNPS consists of a set of neurons placed in the nodes of a directed graph and with the ability of sending spikes along the arcs of the graph (called synapses). In SNPS, the objects, i.e. spikes, evolve by means of *spiking and forgetting rules*. Subsequently, many variants of SNPS were investigated and shortly they become a very attractive and promising branch in the area of membrane computing.[70,71] Among the various SNPS, several language generators were studied in Refs. 16, 19, 62, 63 and 91. The results show that SNPS can generate various languages such as binary strings.

Inspired by the language generative capacity of SNPS, this paper proposes a way to design SNPS for directly solving a combinatorial optimization problem. Unlike all the past studies that combine P systems with evolutionary algorithms, this study is the first attempt to directly derive an optimization algorithm from membrane computing models. More specifically, this paper introduces an extended SNPS (ESNPS), by introducing the probabilistic selection of evolution rules and the output collection from multiple neurons, and further designing a family of ESNPS, called optimization spiking neural P system (OSNPS), through introducing a guider to adaptively adjust rule probabilities. Knapsack problems, a class of well-known NP-complete combinatorial optimization problems, are used as an example to test the optimization capability of OSNPS. A large number of experimental results show that OSNPS has competitive optimization performance with six algorithms reported in recent years.

This paper proposes a design strategy of a neural system that is capable of solving optimization problems. The proposed neural system is a P system that, unlike in MIEAs,[84,87] achieves the optimization results without the aid (in the optimization phase) of a metaheuristic. An ESNPS is developed by introducing the probabilistic selection of evolution rules and multi-neurons outputs and further a family of ESNPS are designed through introducing a guider to adaptively adjust rule probabilities to show how to use ESNPS to approximately solve a single objective and unconstrained combinatorial optimization problems. In other words, an optimization metaheuristic is used only to process the chromosomes by comparing their fitness values and consequently updating the probability values of the SNN and not to generate trial solutions in the optimization phase. In this sense, the optimization is entirely carried out by the spiking neural network. To our knowledge, this is the first study that proposes the use of stand-alone SNPS to tackle optimization problems. The viability of the proposed neural system approach has been tested on knapsack problems, a class of well-known NP-complete combinatorial optimization problems. The choice of this class of problems, at the current prototypical stage, has been carried out since it allows an easy implementation. Furthermore, since

the problem has been intensively studied in the literature, performance comparisons can be straightforwardly done.

The remainder of this paper is organized in the following way. Section 2 briefly introduces SNPS. Section 3 presents the proposed OSNPS in detail. Experiments and results are described in Sec. 4. Concluding remarks are given in Sec. 5 while a short description of the future developments of this work is given in Sec. 6.

2. Spiking Neural P Systems

This section briefly reviews the definition of SNPS as presented in Refs. 40, 52, 70 and 71. In Ref. 61, Păun and Pérez–Jiménez made the description for SNPS. "*SNPS were introduced in Ref.* 40 *in the precise* (*and modest*: *trying to learn a new* "*mathematical game*" *from neurology, not to provide models to it*) *aim of incorporating in membrane computing ideas specific to spiking neurons*; *the intuitive goal was to have* (1) *a tissue-like P system with* (2) *only one* (*type of*) *object*(*s*) *in the cells — the spike, with* (3) *specific rules for evolving populations of spikes, and* (4) *making use of the time as a support of information.*"

A SNPS of degree $m \geq 1$ is a tuple $\Pi = (O, \sigma_1, \ldots, \sigma_m, syn, i_0)$, where:

(1) $O = \{a\}$ is the singleton alphabet (a is called *spike*);
(2) $\sigma_1, \ldots, \sigma_m$ 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:
   (1) $E/a^c \rightarrow a; d$, where E is a regular expression over O, and $c \geq 1$, $d \geq 0$;
   (2) $a^s \rightarrow \lambda$, for some $s \geq 1$, with the restriction that for each rule $E/a^c \rightarrow a; d$ of type (1) 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) $i_0 \in \{1, \ldots, m\}$ indicates the *output* neuron (i.e. σ_{i_0} is the output neuron).

The rules of type (1) are *firing* or *spiking* rules and are used in the following manner: if neuron σ_i contains k spikes, and $a^k \in L(E), k \geq c$, then the rule $E/a^c \rightarrow a; d$ can be applied. The use of this rule means consuming (removing) c spikes (thus only $(k-c)$ spikes remain in neuron σ_i), the neuron is fired, sending a spike out along all outgoing synapses after d time units (in synchronous mode). If $d = 0$, the spike is emitted immediately; if $d = 1$, the spike will be emitted in the next step, etc. If the rule is used at step t and $d \geq 1$, then at steps $t, t+1, \ldots, t+d-1$ the neuron is closed, so that it cannot receive new spikes (if a neuron has a synapse to a closed neuron and tries to send a spike along it, then the particular spike is lost). In the step $t + d$, the neuron spikes becomes open again, so that it can receive spikes (which can be used in step $t + d + 1$). If a rule $E/a^c \rightarrow a; d$ has $E = a^c$, it can be simplified in the form $a^c \rightarrow a; d$. If a rule $E/a^c \rightarrow a; d$ has $d = 0$, it can be written as $E/a^c \rightarrow a$.

The rules of type (2) are *forgetting* rules and they are applied as follows: if neuron σ_i contains exactly s spikes, the rule $a^s \rightarrow \lambda$ from R_i can be applied, indicating that all s spikes are removed from σ_i.

In each time unit, if one of the rules within a neuron σ_i is applicable, a rule from R_i *must* be applied. If two or more rules are available in a neuron, only one of them is chosen in a nondeterministic way. The firing rule and forgetting rule in a neuron are not applicable simultaneously. Thus, the rules are used in the sequential manner in each neuron, but neurons function in parallel with each other.

A configuration of Π at any instant t is a tuple $(n_1, d_1), \ldots, (n_m, d_m)$, where n_i describes the number of spikes present in the neuron σ_i at the instant t and d_i represents the number of steps to count down until it becomes open. The initial configuration of Π is $(n_1, 0), \ldots, (n_m, 0)$, that is, all neurons are open initially. Using the rules of the system in the way described above, a configuration C' can be reached from another configuration C; such a step is called a transition step.

A computation of Π is a (finite or infinite) sequence of configurations such that: (a) the first term of the sequence is the initial configuration of the system and each of the remaining configurations are obtained from the previous one by applying rules of the system in a maximally parallel manner with the restrictions previously mentioned; and (b) if the sequence is finite (called halting computation) then the last term of the sequence is a halting

configuration, that is a configuration where all neurons are open and no rule can be applied to it. With any computation $(C_0, C_1, C_2, \ldots)$ we associate a spike train: the sequence of steps i such that C_i sends a spike out, that is, the sequence of zeros and ones describing the behavior of the output neuron: if the output neuron spikes, we write 1, otherwise we write 0.

3. Optimization Spiking Neural P System

Inspired by the fact that an SNPS is able to generate string languages or spike trains,[16,63] an ESNPS has been designed to produce a binary string, which is used to represent a chromosome or an individual in the description of optimization procedure. The proposed ESNPS introduces the probabilistic selection of evolution rules and collects the output from multiple neurons. Moreover, a novel family of ESNPS obtained by introducing a guider, which is responsible for dealing with a population of chromosomes, to guide the evolution of ESNPS toward the desired output is introduced here.

An ESNPS of degree $m \geq 1$, as shown in Fig. 1, is described as the following construct

$$\Pi = (O, \sigma_1, \ldots, \sigma_{m+2}, \text{syn}, I_0), \tag{2}$$

where:

(1) $O = \{a\}$ is the singleton alphabet (a is called spike);
(2) $\sigma_1, \ldots, \sigma_m$ are neurons of the form $\sigma_i = (1, R_i, P_i), 1 \leq i \leq m$, and $r_i^1 = \{a \to a\}$ and $r_i^2 = \{a \to \lambda\}$, $\sigma_{m+1} = \sigma_{m+2} = (1, \{a \to a\})$, where $R_i = \{r_i^1, r_i^2\}$ is a set of rules of the type and $P_i = \{p_i^1, p_i^2\}$ is a finite set of probabilities, where p_i^1 and p_i^2 are the selection probabilities of rules r_i^1 and r_i^2, respectively, and satisfy $p_i^1 + p_i^2 = 1$.
(3) $\text{syn} = \{(i,j) | (1 \leq i \leq m+1 \wedge j = m+2) \vee (i = m+2 \wedge j = m+1)\}$.
(4) $I_0 = \{1, 2, \ldots, m\}$ is a finite set of *output* neurons, i.e. the output is a spike train formed by concatenating the outputs of $\sigma_1, \sigma_2, \ldots, \sigma_m$.

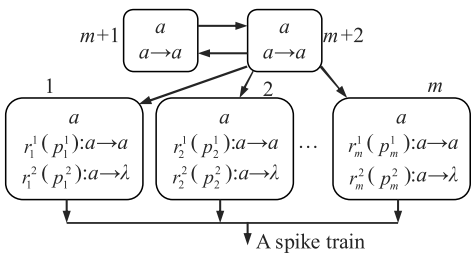

Fig. 1. An example of ESNPS structure.

This system contains the subsystem consisting of neurons σ_{m+1} and σ_{m+2}, which was described in Ref. 63, as a step by step supplier of spikes to neurons $\sigma_1, \ldots, \sigma_m$. In this subsystem, there are two identical neurons, each of which fires at each moment of time and sends a spike to each of neurons $\sigma_1, \ldots, \sigma_m$, and reloads each other continuously. At each time unit, each of neurons $\sigma_1, \ldots, \sigma_m$ performs the firing rule r_i^1 by probability p_i^1 and the forgetting rule r_i^2 by probability p_i^2, $i = 1, 2, \ldots, m$. If the ith neuron spikes, we obtain its output 1, i.e. we obtain 1 by probability p_i^1, otherwise, we obtain its output 0, i.e. we obtain 0 by probability p_i^2, $i = 1, 2, \ldots, m$. Thus, this system outputs a spike train consisting of 0 and 1 at each moment of time. If we can adjust the probabilities $p_1^1, \ldots, p_m^1$, we can control the outputted spike train. In the following paragraphs, a method to adjust the probabilities $p_i^1, \ldots, p_m^1$ by introducing a family of ESNPS is presented.

A certain number of ESNPS can be organized into a family of ESNPS (called OSNPS) by introducing a guider to adjust the selection probabilities of rules inside each neuron of each ESNPS. The structure of OSNPS is shown in Fig. 2, where OSNPS consists of H ESNPS, $\text{ESNPS}_1, \text{ESNPS}_2, \ldots, \text{ESNPS}_H$. Each ESNPS is identical with the one in Fig. 1 and the pseudocode algorithm of the guider algorithm is illustrated in Fig. 3.

The input of the guider is a spike train T_s with $H \times m$ bits and the output is the rule probability matrix $P_R = [p_{ij}^1]_{H \times m}$, which is composed of the rule probabilities of H ESNPS, i.e.

$$P_R = \begin{pmatrix} p_{11}^1 & p_{12}^1 & \cdots & p_{1m}^1 \\ p_{21}^1 & p_{22}^1 & \cdots & p_{2m}^1 \\ \vdots & \vdots & \ddots & \vdots \\ p_{H1}^1 & p_{H2}^1 & \cdots & p_{Hm}^1 \end{pmatrix}. \tag{3}$$

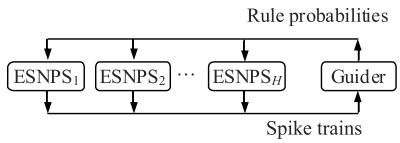

Fig. 2. The proposed OSNPS.

```
Input: Spike train T_s, p_j^a, Δ, H and m
Rearrange T_s as matrix P_R
  i = 1
while (i ≤ H) do
    j=1
   while (j ≤ m) do
     if (rand < p_j^a) then
        k_1, k_2 = ceil(rand * H), k_1 ≠ k_2 ≠ i
        if (f(C_{k_1}) > f(C_{k_2})) then
          b_j = b_{k_1}
        else
          b_j = b_{k_2}
        end if
        if (b_j > 0.5) then
          p_{ij}^1 = p_{ij}^1 + Δ
        else
          p_{ij}^1 = p_{ij}^1 − Δ
        end if
     else
        if (b_j^{max} > 0.5) then
          p_{ij}^1 = p_{ij}^1 + Δ
        else
          p_{ij}^1 = p_{ij}^1 − Δ
        end if
     end if
     if (p_{ij}^1 > 1) then
        p_{ij}^1 = p_{ij}^1 − Δ
     else
        if (p_{ij}^1 < 0) then
          p_{ij}^1 = p_{ij}^1 + Δ
        end if
     end if
     j = j + 1
   end while
   i = i + 1
end while
Output: Rule probability matrix P_R
```

Fig. 3. Guider algorithm.

The guider algorithm in this study is designed for solving a (specific) single objective and unconstrained combinatorial optimization problems. In principle, the guider can also be modified in order to be suitable for other types of optimization problems, such as constrained, multi-objective, numeric optimization problems. However, more work is required in this regard especially to have an efficient coordination between guider and ESNPS.

To clearly understand the guider, we describe its details step by step as follows:

Step 1: Input the learning probabilities $p_j^a, 1 \leq j \leq m$ and the learning rate Δ. Rearrange the input spike train T_s as the rule probability matrix P_R, where each row comes from the identical ESNPS and can be used to represent a chromosome or an individual in an optimization application.

Step 2: Assign the row indicator the initial value $i = 1$.

Step 3: If the row indicator is greater than its maximum H, i.e. $i > H$, the algorithm goes to Step 11.

Step 4: Assign the column indicator the initial value $j = 1$.

Step 5: If the column indicator is greater than its maximum m, i.e. $j > m$, the algorithm goes to Step 10.

Step 6: If a random number rand is less than the prescribed learning probability p_j^a, the guider performs the following two steps, otherwise, it goes to Step 7.

(i) Choose two distinct chromosomes k_1 and k_2 that differs from the ith individual among the H chromosomes, i.e. $k_1 \neq k_2 \neq i$. If $f(C_{k_1}) > f(C_{k_2})$ ($f(\cdot)$ is an evaluation function to an optimization problem; C_{k_1} and C_{k_2} denote the k_1th and k_2th chromosomes, respectively), i.e. the k_1th chromosome is better than the k_2th one in terms of their fitness values (here we consider a maximization problem), the current individual learns from the k_1th chromosome, i.e. $b_j = b_{k_1}$, otherwise, the current individual learns from the k_2th chromosome, i.e. $b_j = b_{k_2}$, where b_j, b_{k_1} and b_{k_2} are intermediate variables, the jth bits of the k_1th and k_2th chromosomes, respectively.

(ii) If $b_j > 0.5$, we increase the current rule probability p_{ij}^1 to $p_{ij}^1 + \Delta$, otherwise, we decrease p_{ij}^1 to $p_{ij}^1 - \Delta$, where Δ is a learning rate.

Step 7: If $b_j^{\max} > 0.5$, the current rule probability p_{ij}^1 is increased to $p_{ij}^1 + \Delta$, otherwise, p_{ij}^1 is decreased to $p_{ij}^1 - \Delta$, where $b_j^{\max}$ is the jth bit of the best chromosome found.

Step 8: If the processed probability p_{ij}^1 goes beyond the upper bound 1, we adjust it to $p_{ij}^1 - \Delta$, otherwise, if the processed probability p_{ij}^1 goes beyond the lower bound 0, we adjust it to $p_{ij}^1 + \Delta$.

Step 9: The column indicator j increases 1 and the guider goes to Step 5.

Step 10: The row indicator i increases 1 and the guider goes to Step 3.

Step 11: The guider outputs the modified rule probability matrix P_R to adjust each probability value of

each evolution rule inside each of neurons $1, \ldots, m$ in each ESNPS.

4. Experimentation and Analysis of Results

To test the feasibility of OSNPS for solving combinatorial optimization problems, this section uses knapsack problems as an application example to conduct experiments. To test the effectiveness of OSNPS for knapsack problems, we consider genetic quantum algorithm (GQA),[31] quantum-inspired evolutionary algorithm (QIEA),[32] novel quantum evolutionary algorithm (NQEA),[20] quantum-inspired evolutionary algorithm (QIEA) based on P systems (QEPS)[81] and two MIEAs with quantum-inspired subalgorithms (MAQIS_1 and MAQIS_2)[86] as benchmark algorithms to carry out comparative experiments and to draw a comparative analysis.

GQA and QIEA are two versions of QIEAs based on the concepts and principles of quantum computing such as quantum-inspired bit, probabilistic observation and quantum-inspired gate.[82] NQEA is an improved QIEA version by modifying the quantum-inspired gate update process. QEPS, MAQIS_1 and MAQIS_2 are three versions of MIEAs. QEPS is based on the use of a P system to properly organize a population of quantum-inspired bit individuals. MAQIS_1 was constructed by using a P system to properly organize five variants of QIEAs based on the consideration that we have no prior knowledge about the performance of the five QIEA variants. MAQIS_2 was designed by using a P system to properly organize QIEA and NQEA based on the investigation in Ref. 90. These approaches represent somehow the state-of-the-art for solving knapsack problems. In order to make the comparison fair, we used both advanced optimization algorithms with a classical approach and modern membrane computing approach. It is a well-known fact, for example, that GQA and QIEA perform better than a classical GA on combinatorial problems of this kind.

4.1. *Knapsack problems*

The knapsack problem, a well-known NP-complete combinatorial optimization problem, can be described as selecting from among various items that are most profitable, given that the knapsack has limited capacity.[25,32] The knapsack problem is to select a subset from the given number of items so as to maximize the profit $f(x)$:

$$f(x) = \sum_{i=1}^{K} p_i x_i \tag{4}$$

subject to

$$\sum_{i=1}^{K} \omega_i x_i \leq C, \tag{5}$$

where K is the number of items; p_i is the profit of the ith item; ω_i is the weight of the ith item; C is the capacity of the given knapsack; and x_i is 0 or 1.

This study uses strongly correlated sets of unsorted data, i.e. the knapsack problem with a linear relationship between the weights and profit values of unsorted items, which were used in Refs. 31–33, 81, 82 and 86 to test the algorithm performance.

$$\omega_i = \text{uniformly random}[1, \Omega], \tag{6}$$

$$p_i = \omega_i + \frac{1}{2}\Omega, \tag{7}$$

where Ω is the upper bound of ω_i, $i = 1, \ldots, K$, and the average knapsack capacity C is applied.

$$C = \frac{1}{2}\sum_{i=1}^{K} \omega_i. \tag{8}$$

4.2. *Analysis of results*

In this subsection, an OSNPS consisting of $H = 50$ ESNPS, each of which has a certain number of neurons such as 1002 for the knapsack problem with 1000 items, is used to solve 11 knapsack problems with respective 1000, 1200, 1400, 1600, 1800, 2000, 2200, 2400, 2600, 2800 and 3000 items. In these problems, $\Omega = 50$ is considered. All the experiments are implemented on the platform MATLAB and on a HP work station with Intel Xeon 2.93 GHz processor, 12GB RAM and Windows 7 OS.

In OSNPS, the learning probability p_j^a ($j = 1, \ldots, m$) and the learning rate Δ are prescribed as a random number in the range [0.05, 0.20] and a random number between 0.005 and 0.02, respectively. In the first three algorithms, GQA, QIEA and NQEA, only one parameter, population size, needs to be set. In the experiments, we set the population size to 50. According to the investigation of QEPS,[81] the population size, the number of elementary membranes and the number of evolutionary generations for the communication of each elementary

membrane are set to 50, 25 and a uniformly random integer ranging from 1 to 10, respectively. The population size and the number of elementary membranes for $MAQIS_1$ are assigned as 50 and 5, respectively. $MAQIS_2$ uses 50 and 2 as the population size and the number of elementary membranes. In the experiments of each algorithm, 30 independent runs are performed for each of the 11 knapsack problems. The stopping condition is prescribed as the number of consecutive generations within which the best solution kept unchanged goes beyond 500, which is useful to exhibit the optimization capability of each algorithm.

In order to handle the case when the total weight of all selected items (fired neurons) exceeds the capacity, we implemented the random chromosome repair technique suggested in Refs. 32, 33 and 82.

The best, worst and average results in terms of maximization of the profit $f(x)$ as in Eq. (4), average generations required for fulfilling an optimization process, and average computing time per generation over 30 independent runs are displayed and listed in Table 1, where the bold style highlights the best result for each problem. More specifically, the BS and WS values (standing for Best Solution and Worst Solution), represent the final objective function values in Eq. (4) of the best and worst run, respectively. The AS value (standing for Average Solution) is the average final objective function values computed over the 30 independent runs available. The values AG and ET (standing for Average Generation and Elapsed Time) represent the average number of evolutionary generations required for fulfilling an optimization process and the average elapsed time per generation (second), respectively; the symbols + and – represent statistical significant difference and no statistical significant difference, respectively.

It is shown in Table 1 that GQA obtains the worst performance among the seven algorithms, in terms of the mean of best, average, worst solutions and average generations. To easily and intuitively show the differences between the seven algorithms, it is appropriate that we choose GQA as a benchmark to draw figures to clearly show how the improvement of each of the other six algorithms is as compared with GQA. Thus, we use the solutions and average generations of GQA as benchmarks to illustrate the percentage of the improvements of QIEA, NQEA, QEPS, OSNPS, $MAQIS_1$ and $MAQIS_2$ in Figs. 4–7. The elapsed time per run of the seven algorithms is shown in Fig. 8.

According to the experimental results, we employ statistical techniques to analyze the behavior of the seven algorithms over the 11 knapsack problems. There are two statistical methods: parametric and nonparametric.[22] The former, also called single-problem analysis, uses a parametric statistical analysis t-test to analyze whether there is a significant difference over one optimization problem between two algorithms. The latter, also called multiple-problem analysis, applies nonparametric statistical tests such as Wilcoxon's and Friedman's tests, to compare different algorithms whose results represent average values for each problem, regardless of the inexistence of relationships among them. Therefore, a 95% confidence Student t-test is first applied to check whether the solutions of the six pairs of algorithms, OSNPS versus GQA, OSNPS versus QIEA, OSNPS versus NQEA, OSNPS versus QEPS, OSNPS versus $MAQIS_1$ and OSNPS versus $MAQIS_2$, are significantly different or not. The results of t-test are also shown in Table 1, where the symbols + and – represent significant difference and no significant difference, respectively. Then two nonparametric tests, Wilcoxon's and Friedman's tests, are employed to check whether there are significant differences between the six pairs of algorithms, OSNPS versus GQA, OSNPS versus QIEA, OSNPS versus NQEA, OSNPS versus QEPS, OSNPS versus $MAQIS_1$ and OSNPS versus $MAQIS_2$. The level of significance considered is 0.05. The results of Wilcoxon's and Friedman's tests are shown in Table 2, where the symbols + and – represent significant difference and no significant difference, respectively.

The experimental results shown in Tables 1 and 2 and Figs. 4–8 indicate the following conclusions:

- OSNPS is superior or competitive to the other six optimization approaches, GQA, QIEA, NQEA, QEPS, $MAQIS_1$ and $MAQIS_2$, with respect to the best, average and worst solutions over 11 problems and 30 independent runs.
- According to the stopping criterion, the more the average generations are, the better balance capability between exploration and exploitation the algorithm has, and as a result the stronger optimization capability the algorithm has. It is shown

Table 1. Experimental results of 11 knapsack problems with $\Omega = 50$ for seven algorithms.

Items		1000	1200	1400	1600	1800	2000	2200	2400	2600	2800	3000
GQA	BS	27,075	32,274	37,034	42,552	47,477	53,151	58,424	62,278	68,239	72,746	78,191
	AS	26,387+	31,769+	36,654+	42,077+	46,934+	52,503+	57,732+	61,733+	67,625+	72,074+	77,628+
	WS	26,128	31,374	36,185	41,594	46,284	51,779	57,198	61,230	67,218	71,610	77,166
	AG	1941	1756	2102	2061	1979	1853	2073	1727	1815	1982	2121
	ET	785	**864**	**1087**	**1244**	**1454**	**1605**	**1808**	**2137**	**2440**	**2740**	**3212**
QIEA	BS	29,729	35,799	41,440	47,478	53,410	59,354	65,324	70,459	77,000	82,536	88,670
	AS	29,533+	35,659+	41,237+	47,175+	53,075+	59,097+	65,029+	70,219+	76,641+	82,081+	88,315+
	WS	29,379	35,549	40,915	46,953	52,810	58,804	64,749	69,909	76,300	81,636	87,845
	AG	3170	3870	4000	4489	5258	5872	6146	7056	7368	7722	8512
	ET	1226	1607	2172	2431	3171	3892	4356	5078	5649	6410	7633
NQEA	BS	29,579	35,699	41,240	47,028	52,810	58,704	64,399	69,308	75,649	80,985	86,570
	AS	29,436+	35,418+	40,958+	46,668+	52,367+	58,102+	63,851+	68,648+	74,790+	79,871+	85,764+
	WS	29,129	35,099	40,515	46,078	51,735	57,378	63,124	67,833	73,323	77,860	85,167
	AG	4342	5117	5886	5530	6497	5932	6653	7132	7106	7789	7351
	ET	1964	2533	3261	3875	4549	5228	5819	6586	7185	7912	8769
QEPS	BS	29,629	35,699	41,340	47,378	53,185	59,479	65,274	70,759	76,850	82,511	88,745
	AS	29,430+	35,527+	41,171+	47,095+	52,926+	59,006+	64,939+	70,239+	76,609+	82,071+	88,321+
	WS	29,179	35,324	40,990	46,853	52,735	58,704	64,474	69,609	76,175	81,661	87,995
	AG	3384	4165	5175	5734	6361	7410	7680	9501	9748	10,495	10,646
	ET	1519	2255	3256	4023	5217	6595	7372	8554	9430	10,650	11,999
OSNPS	BS	**29,979**	**36,174**	**41,865**	**47,927**	**53,809**	**59,854**	**65,899**	**71,307**	**77,622**	**83,109**	**89,458**
	AS	**29,919**	**36,049**	**41,743**	**47,753**	**53,595**	**59,589**	65,660	70,805	77,215	82,588	89,008
	WS	**29,804**	**35,773**	**41,514**	**47,402**	**53,283**	59,178	65,272	70,233	76,448	81,909	88,566
	AG	**7123**	**7406**	**8357**	**9148**	**9544**	**9445**	**11,070**	**12,326**	**12,482**	**12,613**	**15,407**
	ET	1469	1938	2896	4047	5218	6241	7519	8755	9821	10,376	11,950
$MAQIS_1$	BS	29,354	35,399	40,939	46,728	52,610	58,579	64,249	69,333	75,624	80,960	86,945
	AS	29,091+	35,045+	40,619+	46,361+	52,152+	58,046+	63,653+	68,749+	75,059+	80,260+	86,406+
	WS	28,704	34,574	40,240	45,978	51,634	57,529	63,074	67,833	74,250	79,461	85,619
	AG	3978	4569	5708	5645	7127	7799	7413	8571	9426	9644	10,475
	ET	**684**	955	1552	2135	2960	3554	4438	5059	6449	7668	9269
$MAQIS_2$	BS	29,829	36,074	41,690	47,703	53,635	59,754	65,849	71,109	77,600	83,086	89,420
	AS	29,686+	35,878+	41,561+	47,555+	53,486+	59,582–	**65,675**–	**70,846**–	**77,351**+	**82,871**+	**89,221**+
	WS	29,504	35,724	41,415	47,253	53,210	**59,379**	**65,474**	**70,509**	**76,950**	**82,486**	**88,770**
	AG	3510	4193	4503	4798	5621	5977	6791	7084	7493	7809	8472
	ET	1572	2483	3565	4606	5912	7145	8497	9546	10,754	11,771	12,932

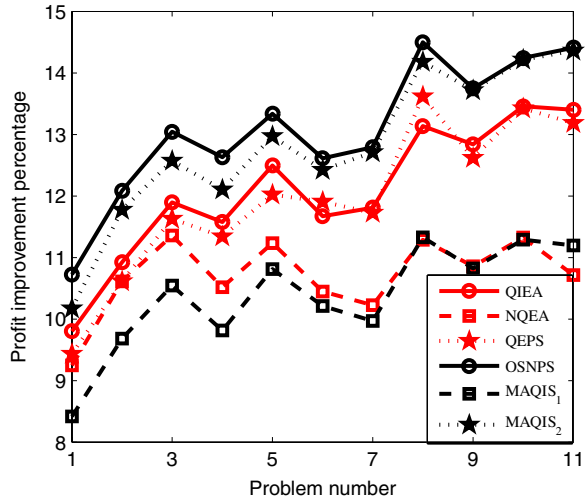

Fig. 4. Maximum profit improvement percentage achieved over the various problems under consideration (best run).

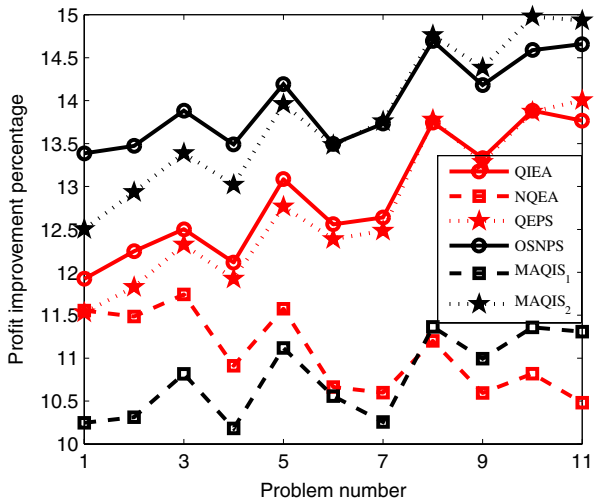

Fig. 5. Average profit improvement percentage achieved over the various problems under consideration.

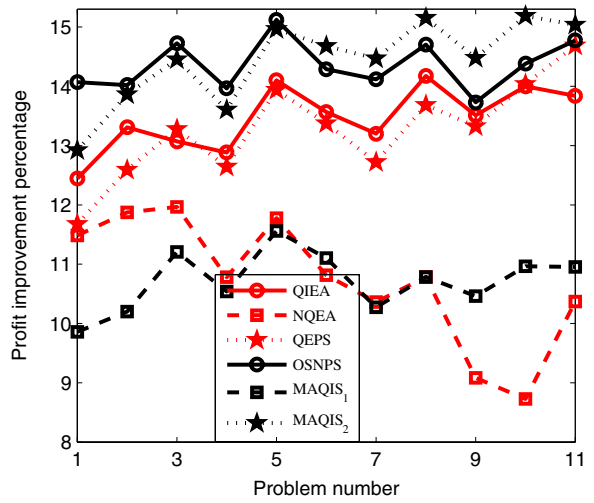

Fig. 6. Minimum profit improvement percentage achieved over the various problems under consideration (worst run).

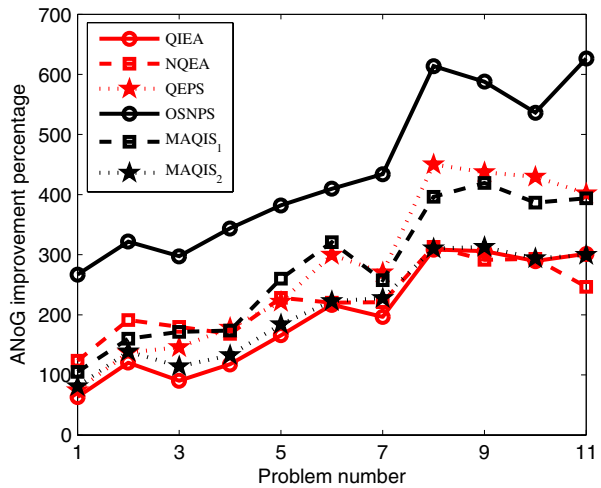

Fig. 7. Average number of generations (ANoG) for an optimization process.

in Fig. 7 that OSNPS is better than the other six optimization approaches in this aspect.

- The three algorithms, QEPS, OSNPS and MAQIS_2, consume more time than the other four approaches, NQEA, MAQIS_1, QIEA and GQA. The elapsed time of QEPS, OSNPS and MAQIS_2 is similar amount. GQA consumes the smallest amount of time.
- The t-test results in Table 1 show that OSNP really outperforms GQA, QIEA, NQEA, QEPS and MAQIS_1 due to 11 significant differences between each of the five pair algorithms, OSNPS versus GQA, OSNPS versus QIEA, OSNPS versus NQEA, OSNPS versus QEPS and OSNPS versus MAQIS_1. OSNPS is really better than MAQIS_2 in 8 out of 11 problems due to eight significant differences and three no significant differences between them.
- The p-values of the two nonparametric tests in Table 2 for the five pair approaches, OSNPS versus GQA, OSNPS versus QIEA, OSNPS versus NQEA, OSNPS versus QEPS and OSNPS versus MAQIS_1, are far smaller than the level of significance 0.05, which indicates that OSNPS

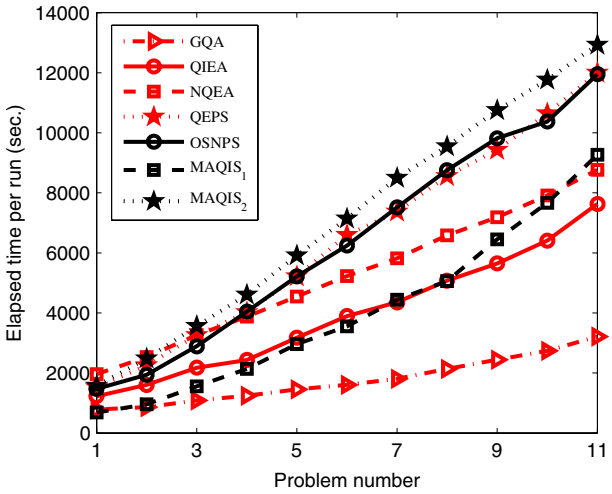

Fig. 8. Elapsed time per run (s) of seven algorithms.

really outperforms GQA, QIEA, NQEA, QEPS and MAQIS_1 over all the 11 problems. The results of Wilcoxon's and Friedman's tests in Table 2 demonstrate that OSNPS is statistically equivalent to MAQIS_2 over the 11 problems because the p-values are greater than 0.05.

4.3. *Statistical ranking by means of Holm–Bonferroni procedure*

In addition to the results presented above, the ranking among all the algorithms considered in this article has been performed by means of the Holm–Bonferroni procedure, see Refs. 21 and 34, for the seven algorithms under study and the 11 problems under consideration. The Holm–Bonferroni procedure consists of the following. Considering the results in the tables above, the seven algorithms under analysis have been ranked on the basis of their average performance calculated over the 11 test problems. More specifically, a score R_i for $i = 1, \ldots, N_A$ (where N_A is the number of algorithms under analysis, $N_A = 7$ in our case) has been assigned. The score has been assigned in the following way: for each problem, a score of 7 is assigned to the algorithm displaying the best performance, 6 is assigned to the second best, 5 to the third and so on. The algorithm displaying the worst performance scores 1. For each algorithm, the scores obtained on each problem are summed up averaged over the amount of test problems (11 in our case). On the basis of these scores, the algorithms are sorted (ranked). With the calculated R_i values, PMS has been taken as a reference algorithm. Indicating with R_0 the rank of PMS, and with R_j for $j = 1, \ldots, N_A - 1$ the rank of one of the remaining 11 algorithms, the values z_j have been calculated as

$$z_j = \frac{R_j - R_0}{\sqrt{\frac{N_A(N_A+1)}{6N_{\text{TP}}}}}, \tag{9}$$

where N_{TP} is the number of test problems in consideration ($N_{\text{TP}} = 11$ in our case). By means of the z_j values, the corresponding cumulative normal distribution values p_j have been calculated. These p_j values have then been compared with the corresponding δ/j where δ is the level of confidence, set to 0.05 in our case. Table 3 displays the ranks, z_j values, p_j values, and corresponding δ/j obtained in this way. The rank of PMS is shown in parenthesis. Moreover, it is indicated whether the null-hypothesis (that the two algorithms have indistinguishable performances) is "Rejected", i.e. PMS statistically outperforms the algorithm under consideration, or "Accepted" if the distribution of values can be considered the same (there is no out-performance).

The Holm–Bonferroni procedure show that the proposed OSNPS displays the highest ranking and that is capable to statistically outperform five of the

Table 2. Results of nonparametric statistical tests, Wilcoxon's and Friedman's tests (WT and FT, for short), for the six pairs of algorithms, OSNPS versus GQA, OSNPS versus QIEA, OSNPS versus NQEA, OSNPS versus QEPS, OSNPS versus MAQIS_1 and OSNPS versus MAQIS_2, in Table 1. The symbols + and – represent significant difference and no significant difference, respectively.

	OSNPS versus GQA	OSNPS versus QIEA	OSNPS versus NQEA	OSNPS versus QEPS	OSNPS versus MAQIS_1	OSNPS versus MAQIS_2
WT	9.7656e-4 (+)	9.7656e-4 (+)	9.7656e-4 (+)	9.7656e-4 (+)	9.7656e-4 (+)	0.7630 (–)
FT	9.1112e-4 (+)	9.1112e-4 (+)	9.1112e-4 (+)	9.1112e-4 (+)	9.1112e-4 (+)	0.8311 (–)

Table 3. Holm test on the fitness, reference algorithm = OSNPS (Rank = 6.54).

j	Optimizer	Rank	z_j	p_j	δ/j	Hypothesis
1	$MAQIS_2$	6.45	–0.09	4.64e-01	5.00e-02	Accepted
2	QIEA	4.81	–1.87	3.01e-02	2.50e-02	Rejected
3	QEPS	4.09	–2.66	3.09e-03	1.67e-02	Rejected
4	NQEA	2.72	–4.14	1.70e-05	1.25e-02	Rejected
5	$MAQIS_1$	2.36	–4.53	1e-06	1.00e-02	Rejected
6	GQA	1	–6.02	1e-06	8.33e-03	Rejected

competitors. Only the $MAQIS_2$ appears to have a performance comparable with that of OSNPS. This result appears very promising considering that in the case of the proposed neural system, the optimization algorithm is designed by a machine and not by a human.

5. Concluding Remarks

This article proposes an effective SNPS design to tackle combinatorial optimization problems. In this study, we proposed a feasible way about how to use SNPS to design an optimization approach for obtaining the approximate solutions of a combinatorial optimization problem. We presented the motivation, algorithmic elaboration and experimental results for verifying the algorithm effectiveness. This work is inspired from language generative SNPS,[16,63] QIEAs,[82] comprehensive learning approaches[44] and estimation of distribution algorithms.[43] Notwithstanding the fact that this work is the first attempt in this direction, the results appear promising and competitive when compared with *ad hoc* optimization algorithms. It must be remarked that this paper starts a new research approach for solving optimization problems. Although more work is required to be competitive with existing optimization algorithms, the clear advantage of the proposed OSNPS is that the optimization algorithm is done by a machine (by a neural system) not by a human designer.

6. Future Work

Future work will attempt to improve upon the optimization performance of the current OSNPS prototype. Other optimization P systems and various applications will also be taken into account. More specific future directions of this research are listed here. *Optimization performance improvement*: on one hand, the performance of OSNPS could be improved by adjusting the parameters such as the learning probability p_a^j and the learning rate Δ. On the other hand, better guiders, to be specific, how to update the rule probabilities, may be devised to enhance the optimization performance of OSNPS. *More combinatorial optimization P systems*: this work presents one way to design a combinatorial optimization P system, so more methods and more P systems could be explored. For instance, inspired by language generative capabilities of numerous P system variants, more variants of OSNPS, optimization cell- and tissue-like P systems might be worthy to be discussed. *Applications*: in this study, knapsack problems were used as examples to test the feasibility and effectiveness of OSNPS, so it is obvious that we can use them to solve various application problems, such as fault diagnosis of electric power systems, robot path planning problems, image segmentation problems, signal and image analysis, power system state estimation including renewable energies, optimization design of controllers for control systems and digital filters, and so on. *Numerical optimization SNPS*: following this work, is it possible to design an optimization SNPS for solving numerical optimization problems by modifying the ingredients of the SNPS? *OSNPS solver*: the OSNPS can be implemented on the platform *P-Lingua*[23,24] or *MeCoSim*[59] and can be developed as an automatic solver for various combinatorial optimization problems.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (61170016, 61373047), the Program for New Century Excellent Talents in University (NCET-11-0715) and SWJTU supported project (SWJTU12CX008).

References

1. H. Adeli and A. Karim, Scheduling/cost optimization and neural dynamics model for construction, *J. Construct. Manag. Eng.* **123**(4) (1997) 450–458.
2. H. Adeli and H. S. Park, *Neurocomputing for Design Automation* (CRC Press, 1998).
3. H. Adeli and H. Park, Method and apparatus for efficient design automation and optimization, and structure produced thereby, US Patent 5,815,394, 29 September 1998.
4. H. Adeli and H. S. Park, A neural dynamics model for structural optimization–theory, *Comput. Struct.* **57**(3) (1995) 383–390.
5. H. Adeli and H. Kim, Cost optimization of composite floors using the neural dynamics model, *Commun. Numer. Meth. Eng.* **17**(11) (2001) 771–787.
6. H. Adeli and A. Panakkat, A probabilistic neural network for earthquake magnitude prediction, *Neural Netw.* **22**(7) (2009) 1018–1024.
7. H. Adeli and H. S. Park, Optimization of space structures by neural dynamics, *Neural Netw.* **8**(5) (1995) 769–781.
8. F. Ahmadkhanlou and H. Adeli, Optimum cost design of reinforced concrete slabs using neural dynamics model, *Eng. Appl. Artif. Intell.* **18**(1) (2005) 65–72.
9. M. Ahmadlou and H. Adeli, Enhanced probabilistic neural network with local decision circles: A robust classifier, *Integr. Comput.-Aided Eng.* **17**(3) (2010) 197–210.
10. F. Alnajjar and K. Murase, Self-organization of spiking neural network that generates autonomous behavior in a real mobile robot, *Int. J. Neural Syst.* **16**(4) (2006) 229–240.
11. A. Belatreche, L. P. Maguire and T. M. McGinnity, Advances in design and application of spiking neural networks, *Soft Comput.* **11**(3) (2007) 239–248.
12. S. M. Bohte and J. N. Kok, Applications of spiking neural networks, *Inform. Process. Lett.* **95**(6) (2005) 519–520.
13. F. Caraffini, F. Neri and L. Picinali, An analysis on separability for memetic computing automatic design, *Inform. Sci.* **265** (2014) 1–22.
14. F. Caraffini, F. Neri, G. Iacca and A. Mol, Parallel memetic structures, *Inform. Sci.* **227** (2013) 60–82.
15. Z. Cen, J. Wei and R. Jiang, A gray-box neural network-based model identification and fault estimation scheme for nonlinear dynamic systems, *Int. J. Neural Syst.* **23**(6) (2013) 1350025.
16. H. Chen, R. Freund, M. Ionescu, G. Păun and M. J. Pérez-Jiménez, On string languages generated by spiking neural P systems, *Fund. Inform.* **75**(1–4) (2007) 141–162.
17. J. Cheng, G. Zhang and X. Zeng, A novel membrane algorithm based on differential evolution for numerical optimization, *Int. J. Unconv. Comput.* **7**(3) (2011) 159–183.
18. S. Ding, H. Li, C. Su, J. Yu and F. Jin, Evolutionary artificial neural networks: A review, *Artif. Intell. Rev.* **39**(3) (2013) 251–260.
19. R. Freund and M. Oswald, Regular omega-languages defined by finite extended spiking neural P systems, *Fund. Inform.* **83**(1–2) (2008) 65–73.
20. H. Gao, G. Xu and Z. R. Wang, A novel quantum evolutionary algorithm and its application, in *Proc. Sixth World Congress on Intelligent Control and Automation* (2006), pp. 3638–3642.
21. S. Garcia, A. Fernandez, J. Luengo and F. Herrera, A study of statistical techniques and performance measures for genetics-based machine learning: Accuracy and interpretability, *Soft Comput.* **13**(10) (2008) 959–977.
22. S. García, D. Molina, M. Lozano and F. Herrera, A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behavior: A case study on the CEC'2005 special session on real parameter optimization, *J. Heuristics* **15**(6) (2009) 617–644.
23. M. García Quismondo, R. Gutiérrez Escudero, M. A. Martinez del Amor, E. Orejuela Pinedo and I. Pérez Hurtado, P-Lingua 2.0: A software framework for cell-like P systems, *Int. J. Comput. Commun. Control* **4**(3) (2009) 234–243.
24. M. García-Quismondo, R. Gutiérrez-Escudero, I. Pérez-Hurtado, M. J. Pérez-Jiménez and A. Riscos-Núñez, An overview of P-lingua 2.0, *Workshop on Membrane Computing*, eds. Gh. Păun, M. J. Pérez-Jiménez, A. Riscos-Núñez, G. Rozenberg and A. Salomaa, Lecture Notes in Computer Science, Vol. 5957 (2010), pp. 264–288.
25. M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness* (W. H. Freeman, 1979).
26. S. Ghosh-Dastidar and H. Adeli, A new supervised learning algorithm for multiple spiking neural networks with application in epilepsy and seizure detection, *Neural Netw.* **22**(10) (2009) 1419–1431.
27. S. Ghosh-Dastidar and H. Adeli, Improved spiking neural networks for EEG classification and epilepsy and seizure detection, *Integr. Comput.-Aided Eng.* **14**(3) (2007) 187–212.
28. S. Ghosh-Dastidar and H. Adeli, Spiking neural networks, *Int. J. Neural Syst.* **19**(4) (2009) 295–308.
29. S. Ghosh-Dastidar, H. Adeli and N. Dadmehr, Mixed-band wavelet-chaos-neural network methodology for epilepsy and epileptic seizure detection, *IEEE Trans. Biomed. Eng.* **54**(9) (2007) 1545–1551.
30. S. Ghosh-Dastidar, H. Adeli and N. Dadmehr, Principal component analysis-enhanced cosine radial basis function neural network for robust epilepsy and seizure detection, *IEEE Trans. Biomed. Eng.* **55**(2) (2008) 512–518.
31. K. H. Han and J. H. Kim, Genetic quantum algorithm and its application to combinatorial

optimization problem, in *Proc. 2000 Congress on Evolutionary Computation* (2000), pp. 1354–1360.
32. K. H. Han and J. H. Kim, Quantum-inspired evolutionary algorithm for a class of combinatorial optimization, *IEEE Trans. Evol. Comput.* **6**(6) (2002) 580–593.
33. K.-H. Han and J.-H. Kim, Quantum-inspired evolutionary algorithms with a new termination criterion, hepsilon gate, and two-phase scheme, *IEEE Trans. Evol. Comput.* **8**(2) (2004) 156–169.
34. S. Holm, A simple sequentially rejective multiple test procedure, *Scand. J. Stat.* **6**(2) (1979) 65–70.
35. L. Huang, X. He, N. Wang and Y. Xie, P systems based multi-objective optimization algorithm, *Prog. Nat. Sci.* **17**(4) (2007) 458–465.
36. L. Huang, I. H. Suh and A. Abraham, Dynamic multi-objective optimization based on membrane computing for control of time-varying unstable plants, *Inform. Sci.* **181**(11) (2011) 2370–2391.
37. G. Iacca, F. Neri, E. Mininno, Y. S. Ong and M. H. Lim, Ockham's razor in memetic computing: Three stage optimal memetic exploration, *Inform. Sci.* **188** (2012) 17–43.
38. G. Iacca, F. Caraffini and F. Neri, Multi-strategy coevolving aging particle optimization, *Int. J. Neural Syst.* **24**(1) (2014) 1450008.
39. J. Iglesias and A. E. P. Villa, Emergence of preferred firing sequences in large spiking neural networks during simulated neuronal development, *Int. J. Neural Syst.* **18**(4) (2008) 267–277.
40. M. Ionescu, G. Păun and T. Yokomori, Spiking neural P systems, *Fund. Inform.* **71**(2–3) (2006) 279–308.
41. S. Johnston, G. Prasad, L. P. Maguire and T. M. McGinnity, An FPGA hardware/software co-design towards evolvable spiking neural networks for robotics application, *Int. J. Neural Syst.* **20**(6) (2010) 447–461.
42. M. Kociecki and H. Adeli, Two-phase genetic algorithm for size optimization of free-form steel space-frame roof structures, *J. Construct. Steel Res.* **90**(6) (2013) 283–296.
43. P. Larrañaga and J. A. Lozano (eds.), *Estimation of Distribution Algorithms*: *A New Tool for Evolutionary Computation* (Kluwer, Boston, MA, 2002).
44. J. J. Liang, A. K. Qin, P. N. Suganthan and S. Baskar, Comprehensive learning particle swarm optimizer for global optimization of multimodal functions, *IEEE Trans. Evol. Comput.* **10**(3) (2006) 281–295.
45. N. R. Luque, J. A. Garrido, J. Ralli, J. J. Laredo and E. Ros, From sensors to spikes: Evolving receptive fields to enhance sensorimotor information in a robot-arm, *Int. J. Neural Syst.* **22**(4) (2012) 1250013.
46. W. Maass, On the computational complexity of networks of spiking neurons, in *NIPS*, eds. G. Tesauro, D. S. Touretzky and T. K. Leen (MIT Press, 1994), pp. 183–190.
47. W. Maass, Lower bounds for the computational power of networks of spiking neurons, *Neural Comput.* **8**(1) (1996) 1–40.
48. W. Maass, Networks of spiking neurons: The third generation of neural network models, *Neural Netw.* **10**(9) (1997) 1659–1671.
49. A. Mohemmed, S. Schliebs, S. Matsuda and N. Kasabov, Span: Spike pattern association neuron for learning spatio-temporal spike patterns, *Int. J. Neural Syst.* **22**(4) (2012) 1250012.
50. E. Nichols, L. J. McDaid and M. N. H. Siddique, Case study on a self-organizing spiking neural network for robot navigation, *Int. J. Neural Syst.* **20**(6) (2010) 501–508.
51. T. Y. Nishida, An application of P systems: A new algorithm for NP-complete optimization problems, in *Proc. 8th World Multi-Conf. Systems, Cybernetics and Informatics* (2004), pp. 109–112.
52. L. Pan and X. Zeng, Small universal spiking neural P systems working in exhaustive mode, *IEEE Trans. Nanobiosci.* **10**(2) (2011) 99–105.
53. A. Panakkat and H. Adeli, Neural network models for earthquake magnitude prediction using multiple seismicity indicators, *Int. J. Neural Syst.* **17**(1) (2007) 13–33.
54. A. Panakkat and H. Adeli, Recurrent neural network for approximate earthquake time and location prediction using multiple seismicity indicators, *Comput.-Aided Civil Infrastruct. Eng.* **24**(4) (2009) 280–292.
55. H. S. Park and H. Adeli, A neural dynamics model for structural optimization–application to plastic design of structures, *Comput. Struct.* **57**(3) (1995) 391–399.
56. H. S. Park and H. Adeli, Distributed neural dynamics algorithms for optimization of large steel structures, *J. Struct. Eng.* **123**(7) (1997) 880–888.
57. G. Păun, Computing with membranes, *J. Comput. Syst. Sci.* **61**(1) (2000) 108–143.
58. G. Păun, G. Rozenberg and A. Salomaa, *The Oxford Handbook of Membrane Computing* (Oxford University Press, Inc., New York, NY, USA, 2010).
59. I. Pérez-Hurtado, L. Valencia-Cabrera, M. J. Pérez-Jiménez, M. A. Colomer and A. Riscos-Núñez, Mecosim: A general purpose software tool for simulating biological phenomena by means of P systems, in *Proc. Int. Conf. Bio-Inspired Computing: Theories and Applications* (2010), pp. 637–643.
60. F. Ponulak and A. Kasinski, Introduction to spiking neural networks: Information processing, learning and applications, *Acta Neurobiol. Exp.* **71**(4) (2011) 409–433.
61. G. Păun and M. J. Pérez-Jiménez, Spiking neural p systems. Recent results, research topics, in

Algorithmic Bioprocesses, eds. A. Condon, D. Harel, J. N. Kok, A. Salomaa and E. Winfree (Springer-Verlag, Berlin Heidelberg, 2009), pp. 273–292.
62. G. Păun and M. J. Pérez-Jiménez, Languages and P systems: Recent developments, *Comput. Sci. J. Moldova* **20**(2) (2012) 112–132.
63. G. Păun, M. J. Pérez-Jiménez and G. Rozenberg, Spike trains in spiking neural P systems, *Int. J. Found. Comput. Sci.* **17** (2006) 975–1002.
64. K. Ramanathan, N. Ning, D. Dhanasekar, G. Li, L. Shi and P. Vadakkepat, Presynaptic learning and memory with a persistent firing neuron and a habituating synapse: A model of short term persistent habituation, *Int. J. Neural Syst.* **22**(4) (2012) 1250015.
65. J. L. Rosselló, V. Canals, A. Morro and A. Oliver, Hardware implementation of stochastic spiking neural networks, *Int. J. Neural Syst.* **22**(4) (2012) 1250014.
66. S. Schliebs, N. Kasabov and M. Defoin-Platel, On the probabilistic optimization of spiking neural networks, *Int. J. Neural Syst.* **20**(6) (2010) 481–500.
67. A. B. Senouci and H. Adeli, Resource scheduling using neural dynamics model of adeli and park, *J. Construct. Manag. Eng.* **127**(1) (1997) 28–34.
68. N. Siddique, L. McDaid, N. Kasabov and B. Widrow, Special issue: Spiking neural networks introduction, *Int. J. Neural Syst.* **20**(6) (2010) v–vii.
69. S. Soltic and N. K. Kasabov, Knowledge extraction from evolving spiking neural networks with rank order population coding, *Int. J. Neural Syst.* **20**(6) (2010) 437–445.
70. T. Song, L. Pan and G. Păun, Asynchronous spiking neural P systems with local synchronization, *Inform. Sci.* **219** (2013) 197–207.
71. T. Song, L. Pan, J. Wang, I. Venkat, K. G. Subramanian and R. Abdullah, Normal forms of spiking neural P systems with anti-spikes, *IEEE Trans. Nanobiosci.* **11**(4) (2012) 352–359.
72. T. J. Strain, L. J. McDaid, T. M. McGinnity, L. P. Maguire and H. M. Sayers, An STDP training algorithm for a spiking neural network with dynamic threshold neurons, *Int. J. Neural Syst.* **20**(6) (2010) 463–480.
73. A. Tashakori and H. Adeli, Optimum design of cold-formed steel space structures using neural dynamics model, *J. Construct. Steel Res.* **58**(12) (2002) 1545–1566.
74. A. K. Vidybida, Testing of information condensation in a model reverberating spiking neural network, *Int. J. Neural Syst.* **21**(3) (2011) 187–198.
75. W. K. Wong, Z. Wang, B. Zhen and S. Y. S. Leung, Relationship between applicability of current-based synapses and uniformity of firing patterns, *Int. J. Neural Syst.* **22**(4) (2012) 1250017.
76. J. Xiao, Y. Jiang, J. He and Z. Cheng, A dynamic membrane evolutionary algorithm for solving DNA sequences design with minimum free energy, *MATCH Commun. Math. Comput. Chem.* **70**(3) (2013) 971–986.
77. J. Xiao, X. Zhang and J. Xu, A membrane evolutionary algorithm for DNA sequence design in DNA computing, *Chin. Sci. Bull.* **57**(6) (2012) 698–706.
78. J. Xiao, Y. Zhang, Z. Cheng, J. He and Y. Niu, A hybrid membrane evolutionary algorithm for solving constrained optimization problems, *Optik* **125**(2) (2014) 897–902.
79. T. Yamanishi, J. Q. Liu and H. Nishimura, Modeling fluctuations in default-mode brain network using a spiking neural network, *Int. J. Neural Syst.* **22**(4) (2012) 1250016.
80. S. Yang and N. Wang, A novel P systems based optimization algorithm for parameter estimation of proton exchange membrane fuel cell model, *Int. J. Hydrogen Energy* **37**(10) (2012) 8465–8476.
81. G. X. Zhang, M. Gheorghe and C. Z. Wu, A quantum-inspired evolutionary algorithm based on P systems for knapsack problem, *Fund. Inform.* **87**(1) (2008) 93–116.
82. G. Zhang, Quantum-inspired evolutionary algorithms: A survey and empirical study, *J. Heuristics* **17**(3) (2011) 303–351.
83. G. Zhang, J. Cheng and M. Gheorghe, A membrane-inspired approximate algorithm for traveling salesman problems, *Rom. J. Inform.Sci. Tech.* **14**(1) (2011) 3–19.
84. G. Zhang, J. Cheng and M. Gheorghe, Dynamic behavior analysis of membrane-inspired evolutionary algorithms, *Int. J. Comput., Commun. Control* **9**(2) (2014) 227–242.
85. G. Zhang, J. Cheng, M. Gheorghe and Q. Meng, A hybrid approach based on differential evolution and tissue membrane systems for solving constrained manufacturing parameter optimization problems, *Appl. Soft Comput.* **13**(3) (2013) 1528–1542.
86. G. Zhang, M. Gheorghe and Y. Li, A membrane algorithm with quantum-inspired subalgorithms and its application to image processing, *Nat. Comput.* **11**(4) (2012) 701–717.
87. G. Zhang, M. Gheorghe, L. Pan and M. J. Pérez-Jiménez, Evolutionary membrane computing: A comprehensive survey and new results, *Inform. Sci.* (2014), Published online, Web: http://dx.doi.org/10.1016/j.ins.2014.04.007
88. G. Zhang, C. Liu and H. Rong, Analyzing radar emitter signals with membrane algorithms, *Math. Comput. Model.* **52**(11–12) (2010) 1997–2010.
89. G. Zhang, F. Zhou, X. Huang, J. Cheng, M. Gheorghe, F. Ipate and R. Lefticaru, A novel membrane algorithm based on particle swarm optimization for

solving broadcasting problems, *J. Univ. Comput. Sci.* **18**(13) (2012) 1821–1841.
90. H. Zhang, G. Zhang, H. Rong and J. Cheng, Comparisons of quantum rotation gates in quantum-inspired evolutionary algorithms, in *Proc. Int. Conf. Natural Computation* (IEEE, 2010), pp. 2306–2310.
91. X. Zhang, X. Zeng and L. Pan, On languages generated by asynchronous spiking neural P systems, *Theor. Comput. Sci.* **410**(26) (2009) 2478–2488.