An Adaptive Optimization Spiking Neural P System for Binary Problems

MING ZHU, QIANG YANG
School of Control Engineering,
Chengdu University of Information Technology, Chengdu, 610225, China
E-mail: zhuming@cuit.edu.cn, yuxia2008@126.com

JIANGPENG DONG, GEXIANG ZHANG
College of Information Science and Technology,
Chengdu University of Technology, Chengdu 610059, China
E-mail: master_djp@163.com, zhxgxdylan@126.com

XUANTAI GOU∗, HAINA RONG, PRITHWINEEL PAUL
School of Electrical Engineering,
Southwest Jiaotong University, Chengdu, 610031, China
E-mail: 491098063@qq.com, ronghaina@126.com, prithwineelpaul@gmail.com

FERRANTE NERI∗
COL Laboratory, School of Computer Science,
University of Nottingham, Nottingham, UK
E-mail: Ferrante.Neri@nottingham.ac.uk

Optimization Spiking Neural P System (OSNPS) is the first membrane computing model to directly derive an approximate solution of combinatorial problems with a specific reference to the 0/1 knapsack problem. OSNPS is composed of a family of parallel Spiking Neural P Systems (SNPS) that generate candidate solutions of the binary combinatorial problem and a Guider algorithm that adjusts the spiking probabilities of the neurons of the P systems. Although OSNPS is a pioneering structure in membrane computing optimization, its performance is competitive with that of modern and sophisticated metaheuristics for the knapsack problem only in low dimensional cases.

In order to overcome the limitations of OSNPS, the present paper proposes a novel Dynamic Guider algorithm which employs an adaptive learning and a diversity based adaptation to control its moving operators. The resulting novel membrane computing model for optimization is here named Adaptive Optimization Spiking Neural P System (AOSNPS). Numerical result shows that the proposed approach is effective to solve the 0/1 knapsack problems and outperforms multiple various algorithms proposed in the literature to solve the same class of problems even for a large number of items (high dimensionality). Furthermore, case studies show that a AOSNPS is effective in fault sections estimation of power systems in different types of fault cases: including a single fault, multiple faults and multiple faults with incomplete and uncertain information in the IEEE 39 bus system and IEEE 118 bus system.

Keywords: Spiking neural system; adaptive optimization spiking neural P system; adaptive learning rate; adaptive mutation; power system fault diagnosis; combinatorial optimization; membrane computing.

1. Introduction

Artificial neural networks (ANNs), also referred to as neural networks (NNs) or connection models, are computational models consisting of interconnected neurons that mimic the behavioral characteristics of biological neural networks and
perform distributed parallel information processing. ANNs have been extensively investigated and widely used in various fields, such as signal and image processing \cite{13,15,52}, classification \cite{14}, pattern recognition \cite{7,37}, earthquake prediction \cite{1,51,52}, epilepsy and seizure detection \cite{27,28}, medical diagnostics \cite{47} and optimization \cite{2,5,12,11}.

In the last decades, ANNs have passed three developmental generations with notable characteristics.

- The fundamental feature of the first generation is McCulloch–Pitts neurons \cite{55} (perceptrons or threshold gates), where only Boolean functions \cite{41} can be processed and output digital results.
- The salient feature of the second generation is the presence of an activation function with weighted learning ability and the fact that analog input and output can be processed \cite{41}.
- The introduction of time \cite{18} (single action spiking of the neuron) concept when neurons encode information is a prominent feature of the third generation of ANNs.

Spiking Neural Networks (SNNs) present a successful ANN structure belonging to the second \cite{23,56,60,26,24} and third generation \cite{23,56}. The success of SNNs is due to the fact that they are both computationally powerful and biologically plausible models of neuronal processing \cite{53,16,8,45,79}. Some successful implementations of SNN are reported in Ref. \cite{42,57,20,22,80} while a successful implementation in the medical domain is presented in Ref. \cite{25}.

Over the past two decades, in parallel with the development of ANNs, other computational devices attracted attention. Specifically, Membrane Computing \cite{54} increased significantly its popularity, gaining relevance as a computational paradigm. Membrane computing is a branch of natural computing which designs computing models, called P systems, inspired from the structure and the functioning of the biological cells. Most of P systems can be viewed as a general purpose automaton equivalent to the Turing Machine \cite{24}. Due to their general purpose nature, P Systems can be used in a broad range of real-world applications, such as computer graphic \cite{21}, cryptography \cite{23} and robot control \cite{8,64}.

Membrane computing models can be divided into three categories on the basis of their structure:

- **cell-like P systems**, inspired by the intracellular communication \cite{24}.
- **tissue-like P systems**, inspired by intercellular communication \cite{14}.
- **neural-like P systems**, inspired by the communication among neurons \cite{44}, just like an ANN.

Hence, at the conceptual level, neural-like P systems can be viewed as an intersection between ANN and Membrane Computing. Although the conceptual definitions of neural P systems have been given in Ref. \cite{44}, one of their greatest success stories leading to practical use is due to the Spiking Neural P Systems (SNPS) \cite{55} which integrate the SNN logic within Membrane Computing. Unlike the other P systems \cite{14,21,74} which use multiple symbols to encode the pieces of information required, SNPS \cite{55,50,71} usually deal with a unique symbol, namely spike, that is repeated multiple times, see also Ref. \cite{49,34}. The name “spike” is due to the analogy with electrical impulses sent from a neuron to another one along the synapses. Among the various membrane computing models, SNPS is one of the most promising and studied \cite{9,48,59,58}. The SNPS consists of a set of neurons placed in the nodes of a graph and sending signals (spikes) along synapses (edges of the graph), under the control of firing and forgetting rules \cite{68,69,70,73}. One neuron is designated as the output neuron of the system and its spikes can be sent out into the environment, thus producing a different interpretations of the result, including: 1) the number of spikes sent out into the environment in total, along the computation; 2) the number of steps between the first two spikes sent to the environment; 3) a so-called spike-train, as the sequence of spikes sent out all along the computation (in every step, 0 or more spikes are sent out into the environment.).

This paper focuses on SNPS and in particular on its use for addressing optimization problems. The first study of solving combinatorial optimization problems by means of SNPS was proposed in Ref. \cite{78} where the 0/1 Knapsack Problem was ad-
addressed. In Ref. 78 an Extended SNPS (ESNPS) was obtained by introducing two additional neurons, the probabilistic selection of evolution rules and the output (spike train) collection from multiple neurons. In such work, multiple ESNPS are arranged in parallel, each of them generating a candidate solution. Furthermore, in Ref. 78 a Guider algorithm has been introduced to adjust probabilities of the spiking rules (by continuous learning generation by generation). The resulting computational model, namely Optimization Spiking Neural P System (OSNPS), appeared to be promising and displayed a performance competitive in comparison with that of six optimization algorithms taken as a reference.

As indicated in Ref. 78 one of the main limitations of OSNPS lies in the fact that the learning rate is constant, while the search may require a different learning rate in different moments of the search. Due to this limitation, it was observed in Ref. 78 that the probability variation operated by the Guider algorithm does not always allow an effective learning of the family of SNPS. Thus the performance of the OSNPS is not as good as that of some modern metaheuristics, especially when a large number of items is considered in the knapsack problem. For the sake of clarity, it must be remarked that unlike membrane-inspired evolutionary algorithms (MIEAs) 32 38 76 77 that combine a P system framework with metaheuristic algorithms to solve optimization problems, OSNPS uses only SNPS to generate the candidate solutions of the combinatorial optimization problems.

The present paper extends the work of Ref. 78 and proposes a novel algorithm, namely Adaptive Optimization Spiking Neural P System (AOSNPS) to solve combinatorial problems with binary encoding. More specifically, a novel adaptive learning rate and adaptive mutation are proposed and integrated within OSNPS. In addition, the proposed AOSNPS includes a mutation operator governed by two dynamic parameters that estimate the accuracy of the candidate solution and diversity of probability matrices. The proposed mutation enhances upon the balance between exploration and exploitation features of OSNPS. Finally, the real application of AOSNPS to fault section estimation of power systems is effective in fault sections estimation of power systems in different types of fault cases including single fault, multiple faults and multiple faults with incomplete and uncertain information.

The remainder of this study is organized as follows: Section 2 briefly introduces the used notation, the basics of SNPS theory used in this article as well as the OSNPS as in its original implementation. Section 3 presents the proposed AOSNPS consisting of adaptive learning rate, adaptive mutation and the novel guider. The numerical results obtained by the algorithm suggested when applied to the 0/1 knapsack problem are shown in Section 4, while an application to power systems, is given in Section 5. Finally, conclusion and future work are given in Section 6.

2. Background: Problem and Optimization by Spiking Neural P Systems

In this section, we briefly introduce the notation and analyse the basic concepts of optimization by SNPS. We address the minimization/maximization of a function \( f(x) \) where \( x = (x_1, x_2, \ldots, x_m) \) is a vector of binary numbers.

The SNPS is an automaton. The distinguishing feature of SNPS is that the sequence of configurations can associate a spike train \( \sigma m,q \) \( m,q \). If the output neuron spikes, then we have 1 and otherwise we have 0. Hence, the spike train can be represented by the sequence of ones and zeros. Here, we give an extended definition of SNPS.

2.1. Extended Spiking Neural P System

Formally, an extended spiking neural P system (abbreviated as ESNPS), of degree \((m, q)\) with \( m, q \geq 1 \), is a tuple of the form

\[
\Pi = (O, \sigma_1, \ldots, \sigma_m, \sigma_{m+1}, \ldots, \sigma_{m+q}, \sigma_{m+q+1}, s)_{\text{out}}
\]

where:

(1) \( O = \{\sigma\} \) is the singleton alphabet (\( \sigma \) is called spike);

(2) \( \sigma_i, 1 \leq i \leq m + q \), is an ordered term \((n_i, R_i, P_i)\) such that:
of a directed graph $\Pi$.

- $R_i$, for $1 \leq i \leq m$, is a finite set of rules of the types:
  
  (i) $E/a^c \rightarrow a; d$, where $E$ is a regular expression over $O$, $c \geq 1$, and $d \geq 0$, $d$ is the delay associated with the rule (spiking rule);

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

- $R_{m+1}, \ldots, R_{m+q}$ is a finite set of spiking rules.

- $P_i$, for $1 \leq i \leq m$, is a map from the set $R_i$ into $[0, 1]$ such that $\sum_{\sigma \in R_i} P_i(\sigma) = 1$.

- $P_{m+1}, \ldots, P_q$ are the identically one functions.

- $\text{syn} \subseteq \{(\sigma_i, \sigma_j) | 1 \leq i, j \leq m + q \land i \neq j\}$.

- $I_{out} \subseteq \{\sigma_1, \ldots, \sigma_m\}$.

An ESNPS, $\Pi$, of degree $(m, q)$, with $m, q \geq 1$ can be viewed as a set of neurons $\{\sigma_1, \ldots, \sigma_m, \sigma_{m+1}, \ldots, \sigma_{m+q}\}$ placed in the nodes of a directed graph $G_\Pi$, where the set $\text{syn}$ specifies the arcs of the graph which are labeled by natural numbers representing the delays associated with synapses: $D_{\text{syn}}(\sigma_i, \sigma_j)$ will denote the label associated with the arc $(\sigma_i, \sigma_j)$. If $(\sigma_i, \sigma_j) \in \text{syn}$ then we say that $\sigma_i$ is a pre-synaptic neuron of $\sigma_j$ or $\sigma_j$ is a postsynaptic neuron of $\sigma_i$. In these systems there are two kind of neurons: $\sigma_1, \ldots, \sigma_m$ are called working neurons, some of them can also act as output neurons, whereas $\sigma_{m+1}, \ldots, \sigma_{m+q}$ are supplier neurons that can send spikes to each working neuron. For each neuron $\sigma_i, n_i$ is the number of spikes initially contained in that neuron, $R_i$ is a finite set of (spiking and/or forgetting) rules with probabilities associated by means of functions $P_i$.

The semantics of such an ESNPS, $\Pi$, is defined as follows. A configuration $C_t$ at a moment $t$ (a natural number, $t \geq 0$) of an ESNPS of degree $(m, q)$ is a tuple $(p_1, \ldots, p_{m+q})$ where $p_i, 1 \leq i \leq m + q$, describes the number of spikes present in the neuron $\sigma_i$ at instant $t$. The initial configuration of $\Pi$, that is, the configuration at moment $t = 0$, is $(n_1, \ldots, n_{m+q})$, being $\sigma_i = (n_i, R_i, P_i)$, for $1 \leq i \leq m + q$.

A spiking rule $E/a^c \rightarrow a; d \in R_i$ is applicable to a configuration $C_t$, at the moment $t$, if the neuron $\sigma_i$ contains exactly $k \geq c$ spikes at instant $t$ and $a^s \notin L(E)$. By applying such a rule, $c$ spikes are consumed from the neuron $\sigma_i$, it is fired and all postsynaptic neurons $\sigma_j$ from the neuron $\sigma_i$ would receive $p$ spikes from neuron $\sigma_i$, at step $t + D_{\text{syn}}(i, j) + 1$, that is, if $D_{\text{syn}}(i, j) = 0$, then the spikes are received by such postsynaptic neuron $\sigma_j$ immediately, otherwise the spikes are received after $D_{\text{syn}}(i, j)$ steps. A forgetting rule $a^s \rightarrow \lambda \in R_i$ is applicable to a configuration $C_t$ at the moment $t$ if the neuron $\sigma_i$ contains exactly $s$ spikes at instant $t$. By applying such a rule, $s$ spikes are removed from the neuron $\sigma_i$.

In each time unit, if neuron $\sigma_i$ is enabled (i.e., one of its rules can be used), then one rule from $R_i$ (and only one) must be applied, chosen non-deterministically among all possible rules applicable to $\sigma_i$. That is, each neuron processes sequentially its spikes by using, at most, one rule in each time unit. Let us recall that a spiking rule and a forgetting rule cannot be simultaneously applied to a neuron. The rules are used in a maximally parallel way at the level of the system: at each step, all neurons which can use some rule, must evolve using one rule.

Using the rules of the system in the way described before, a configuration $C_{t+1}$ can be reached from another configuration $C_t$; such a step is called a transition step, and we denote it by $C_t \Rightarrow C_{t+1}$. Any (finite or infinite) sequence of transitions $C_0 \Rightarrow C_1 \Rightarrow C_2 \Rightarrow \ldots \Rightarrow C_r$, $r \in \mathbb{N} \cup \{\infty\}$, is a computation if the following holds: (a) the first term of the sequence, $C_0$, is the initial configuration of the system; (b) for each $n$, the $(n+1)$-th term of the sequence is obtained from the $n$-th term in one transition step; and (c) if the sequence is finite (called halting computation) then the last term $C_r$, $r \in \mathbb{N}$, of the sequence is called a halting configuration, that is, a configuration where no rule of the system is applicable to it.

In order to propose a novel algorithm, AOS-NSP, to solve combinatorial optimization problems, a specific ESNPS (where $q = 2$ and $n_i = 1$), is considered with the following syntactical restrictions:

$$\Pi = (O, \sigma_1, \ldots, \sigma_m, \sigma_{m+1}, \sigma_{m+2}, \text{syn}, I_0),$$

where

(1) $O = \{a\}$ is the singleton alphabet, $a$ is called spike
(2) \( \sigma_1, \ldots, \sigma_m \) are neurons of the form
\[ \sigma_i = (1, R_i, P_i). \]

Further comments on the neurons are:
(a) In each neuron \( \sigma_i \), there is only 1 initial spike, i.e., \( n_i = 1 \), for \( i = 1, 2, \ldots, m + 2 \).
(b) \( R_i = \{ r_i^1, r_i^2 \} \) is a set of rules, spiking \( r_i^1 : a \rightarrow a \) and forgetting \( r_i^2 : a \rightarrow \lambda \), respectively.
(c) \( P_i = \{ p_i^1, p_i^2 \} \) 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) The two additional neurons, \( \sigma_{m+1} = \sigma_{m+2} = (1, a \rightarrow a) \), work as a step by step supplier of spikes to neurons \( \sigma_1, \sigma_2, \ldots, \sigma_m \).

(4) \( \text{syn} = \{(\sigma_i, \sigma_j) | (i = m + 2 \text{ AND } 1 \leq j \leq m + 1 \text{ OR } (i = m + 1 \text{ AND } j = m + 2) \} \).

(5) \( \text{lo} = \{1, 2, \ldots, m\} \) is a finite set of output neurons, i.e., the output is a spike train formed by concatenating the outputs of \( \sigma_1, \sigma_2, \ldots, \sigma_m \).

The structure of the specific ESNPS is shown in Fig. 1.

![Figure 1. Logical and functioning scheme of the specific ESNPS](image)

2.2. Optimization Spiking Neural P System

The OSNPS is composed of multiple parallel ESNPSs and a Guider algorithm, that supervises the family of ESNPSs, as shown in Fig. 2.

![Figure 2. Structure of OSNPS (and AOSNPS)](image)

Each ESNPS, with \( 1 \leq i \leq H \), has the structure shown in Fig. 1. The Guider algorithm in Fig. 2 is used to adjust the probabilities \( P_i \) of each ESNPS. The input of the Guider algorithm is a spike train \( T_i \), that is a set of \( H \) binary strings/candidate solutions of length \( m \) (a binary matrix of \( H \times m \) elements) produced by each ESNPS. The output of the Guider algorithm is the rule probability matrix \( P_R = [P_{ij}]_{H \times m} \) (the apex 1 indicates the probability of the bit to be 1), which is composed of the spiking rule probabilities of \( H \) ESNPSs, i.e.,

\[
P_R = \begin{pmatrix}
p_{11} & p_{12} & \cdots & p_{1m} \\
p_{21} & p_{22} & \cdots & p_{2m} \\
\vdots & \vdots & \ddots & \vdots \\
p_{H1} & p_{H2} & \cdots & p_{Hm}
\end{pmatrix}
\]

The probabilities are fed back to the ESNPSs which are updated accordingly.

![Figure 3. The Guider algorithm of OSNPS](image)
Fig. 3 provides the details of the Guider algorithm used in OSNPS [78]. The basic idea is that one current best solution (binary vector)
\[ x = (x_1, x_2, \ldots, x_m) \]
is stored in memory and two candidate solutions \( x_k1 \) and \( x_k2 \) are randomly selected, with a certain probability \( p_{ij}^k \), and compared on the basis of a fitness (objective function) \( f \). If appropriate, the constraints are handled.

If \( x_k1 \) and \( x_k2 \) are generated, for each design variable \( x_j \), the current solution is perturbed (\( x \) inherits the design variable of the winning candidate solution). Then, the probability associated with the most successful solution is increased by a constant learning rate \( \Delta \). Conversely, the probability associated with the candidate solution displaying a lower performance is reduced. If \( x_k1 \) and \( x_k2 \) are not generated, the binary variable of \( x_j \) is directly used to affect the probability, see lines 19-24 of the pseudocode in Fig. 3. More details on OSNPS can be found in Ref. 78.

3. Adaptive Optimization Spiking Neural P System

The proposed AOSNPS employs the same structure of OSNPS depicted in Fig. 2 but makes use of a novel Dynamic Guider algorithm to manipulate the probability matrix \( P_R \). More specifically, the Guider algorithm contains two novel adaptive mechanisms: 1) an adaptive learning rate \( \Delta \); 2) an adaptive mutation. The novel mechanisms are described in Subsections 3.1 and 3.2, respectively. The new Dynamic Guider algorithm embedding these two elements is outlined in Subsection 3.3.

3.1. Adaptive Learning Rate

The learning rate \( \Delta \) is the step size of probability adjustment for the elements of the probability matrix \( P_R \):
\[
\begin{align*}
p_{ij}^1 &= p_{ij}^1 + \Delta \\
p_{ij}^1 &= p_{ij}^1 - \Delta
\end{align*}
\]
with \( 1 \leq i \leq H \) and \( 1 \leq j \leq m \). In Ref. 78, \( \Delta \) is a random number between 0.005 and 0.02 set at the beginning of the optimization and kept constant throughout the OSNPS execution.

We propose here a variable and adaptive learning rate \( \Delta \). In order to better explain the rationale of the proposed adaptation, let us revisit the role of \( p_{ij}^1 \) of ESNPS presented in Section ??.

The term \( p_{ij}^1 \) is the selection probability of rule \( r_i^1 \) while \( r_i^1 \) is the spiking rule in each neuron. If the value of \( p_{ij}^1 \) is large, the rule \( r_i^1 \) has a high probability of execution and the rule \( r_i^1 \) has a low probability of execution since \( p_{ij}^1 + p_{ij}^2 = 1 \). If the output neuron spikes, a 1 is written as a bit of the candidate solution, otherwise a 0 is written. Thus, if we want to get 1, \( p_{ij}^1 \) should be large (ideally \( p_{ij}^1 = 1 \)) and if we want to get 0, \( p_{ij}^1 \) should be small (ideally \( p_{ij}^1 = 0 \)).

This paper proposes an adaptive probability adjustment step size for each neuron. At each time unit, the adaptive updating rule of probability is
\[
p_{ij}^1 = p_{ij}^1 + \Delta_{ij}^a
\]
where \( \Delta_{ij}^a \) (\( a \) stands for “adaptive”) is the step size and is defined as
\[
\Delta_{ij}^a = \frac{P_b - p_{ij}^1}{2}.
\]
\( \Delta_{ij}^a \) is designed to take the middle point of the distance between the current probability \( p_{ij}^1 \) and the ideal probability. The ideal probability \( P_b = \{0, 1\} \) is the lower or upper bound of the probability of \( p_{ij}^1 \) (the pedge \( b \) stands for “bound”). For \( P_b = 1 \), the update rule is
\[
p_{ij}^1 = p_{ij}^1 + \frac{1 - p_{ij}^1}{2} = 0.5 + 0.5p_{ij}^1
\]
while for \( P_b = 0 \) the update rule is
\[
p_{ij}^1 = p_{ij}^1 + \frac{0 - p_{ij}^1}{2} = 0.5p_{ij}^1.
\]

Compared with the learning rate \( \Delta \) defined in OSNPS, the adaptive learning rate \( \Delta_{ij}^a \) proposed in this paper presents the following advantages.

- \( \Delta_{ij}^a \) changes for each neuron at each time unit during the algorithm execution following the learning needs. If the distance between the current probability \( p_{ij}^1 \) and the ideal probability \( P_b \) is big, the learning rate \( \Delta_{ij}^a \) is big. On the other hand, if the distance between the current probability \( p_{ij}^1 \) to the ideal probability \( P_b \) is small, the learning rate is also small.
- The adaptive learning rate allows a quick achievement of the desired probability. For example, if we want to get 1 from a neuron, from an initial probability $p_{ij}^{(0)} = 0.1$ the proposed adaptive learning rate $\Delta_t^a$ enables to reach $0.9 < p_{ij}^{(t)} < 1$ after four steps whereas with a constant $\Delta_t$ over 40 steps are needed.

- The probability of $p_{ij}^{(t)}$ does not overflow. In OSNPS, it may result $p_{ij}^{(t)} > 0$ or $p_{ij}^{(t)} < 0$. In this case, an extra mechanism is required. The use of the proposed adaptive learning rate ensures that the probability overflow never occurs.

3.2. Adaptive Mutation

The proposed AOSNPS makes also use of an adaptive mutation strategy. Two dynamic parameters $P_{m_1}$ and $P_{m_2}$ are defined to characterise and monitor the learning (evolutionary state) of AOSNPS. The parameter $P_{m_1}$ is the first mutation probability which varies between 0 and 1 and is defined as follows:

\[
\text{if } G_{bf}(\text{gen}) > G_{bf}(\text{gen} - 1) \text{ then } P_{m_1} = 0 \text{ end if}
\]

\[
\text{if } G_{bf}(\text{gen}) = G_{bf}(\text{gen} - 1) \text{ then } P_{m_1} = P_{m_1} + \frac{1}{N_{\max}} \text{ end if}
\]

where $G_{bf}(\text{gen})$ is the best fitness value ever computed at the generation time of the execution and $G_{bf}(0)$ is the best fitness at the initialization. The parameter $N_{\max}$ determines a stopping criterion: if the global best fitness does not improve for consecutive $N_{\max}$ generations the algorithm stops.

The parameter $P_{m_2}$ is the second mutation probability and is defined as follows:

\[
P_{m_2} = \frac{DP_a(\text{gen})}{DP_a(0)}
\]

where $DP_a(\text{gen})$ is an aggregated metric representing the diversity of probabilities at the current generation $\text{gen}$, see [40]. This value is calculated as:

\[
DP_a(\text{gen}) = \frac{2}{H \times M \times N} \sum_{i=1}^{H-1} \left( \sum_{j=i+1}^{H} \frac{1}{M} \sum_{k=1}^{m} \left| p_{ik}^{\text{gen}} - p_{jk}^{\text{gen}} \right| \right).
\]

The value $DP_a(0)$ is the average probability at the initialization.

The triggering rule for adaptive mutation is defined as

\[
\text{rand}_1() < P_{m_1} \text{ AND } \text{rand}_2()> P_{m_2}
\]

where $\text{rand}_1()$ and $\text{rand}_2()$ are two random numbers in the range $[0, 1]$.

$P_{m_1}$ and $P_{m_2}$ are two independent dynamic parameters. $P_{m_1}$ is used to describe the trend of improvement of the optimal solution. $P_{m_2}$ is used to describe the diversity of probability sample matrix between rows in the current generation. $N_{\max}$ is the parameter of when the algorithm should stop running. The value of $N_{\max}$ affects the standard deviation between the results of multiple runs of AOSNPS for the same issue. In order to make the results of multiple runs more concentrated, we should choose a larger $N_{\max}$. The logic of the mutation probabilities $P_{m_1}$ and $P_{m_2}$ is summarised in the following points.

- If a new current best solution has been detected at the current generation ($G_{bf}(\text{gen}) > G_{bf}(\text{gen} - 1)$), then the mutation is not triggered ($P_{m_1} = 0$). Conversely, if the current best solution has not been updated ($G_{bf}(\text{gen}) = G_{bf}(\text{gen} - 1)$), the probability of triggering the mutation is increased ($P_{m_1} = P_{m_1} + \frac{1}{N_{\max}}$).

- If the diversity $DP_a(\text{gen})$ is larger than that at initialization $DP_a(0)$, the probability for the mutation to be triggered is low.

- If the current best solution is not updated for many generations ($G_{bf}(\cdot)$ remains the same), and the diversity $DP_a(\text{gen})$ of the probability matrix $P_R$ is low, the mutation is performed with a high probability ($P_{m_1}$ is large, $P_{m_2}$ is small).

When triggered the mutation of the matrix $P_R$ is performed in the following way:

\[
\text{for } i = 1, 2, \ldots, H \text{ with } i \neq R_{\text{bestfit}} \text{ do}
\]

\[
\text{for } j = 1, 2, \ldots, m \text{ do}
\]

\[
\text{if } \text{rand}_3() < P_{m_2} \text{ then }
\]

\[
p_{ij}^{(t)} = \text{rand}()
\]

end if

end for

end for

where $\text{rand}_3()$ and $\text{rand}()$ are two random numbers
in the range [0, 1]. The mutation probability $P^{m}_{j}$ is a parameter sampled in the range [0, 0.1] at the beginning of the Dynamic Guider algorithm’s execution. $R_{bestfit}$ ∈ [1, H] is the row coordinate of the best candidate solution found at the current generation. This means that the best candidate solution, in an elitist fashion, does not participate in the mutation.

Based on the adaptive learning rate and the adaptive mutation rule, the Dynamic Guider algorithm is an algorithm operating on the learning probabilities to support the generation of successful solution for the binary combinatorial problem.

One current best solution $x$ is stored in memory and modified by crossover with other candidate solutions. The candidate solutions are generated by the parallel ESNPSs (one solution by each ESNPS$_j$) and processed in pairs by the evolutionary Dynamic Guider algorithm that evaluates and compare their fitness values. The evaluation of these candidate solutions are used to modify the probability matrix $P_R$ and hence generate solutions with a higher performance. Like for OSNPS, the constraint is handled by means of the random chromosome repair technique, see 29 30 75.

The evolution of the $P_R$ matrix is handled by two mechanisms: the update of the adaptive learning rate which happens every time two candidate solutions are sampled as described in Subsection 3.1 the re-initialisation of the neuron probability which is triggered only by the satisfaction of the conditions described in Subsection 3.2.

3.3. The Dynamic Guider Algorithm of AOSNPS

Hence, we may consider AOSNPS as a co-evolution where the (binary) candidate solutions of the optimization problem evolve with the probabilities of the neurons in each of the P systems. The pseudocode of the proposed evolutionary Dynamic Guider is shown in Fig. 4.

4. Numerical Results: 0/1 knapsack problem

Given a group of items, each item with its own weight and price, and a knapsack of limited capacity, the problem consists of selecting the items to make the total price of the knapsack as high as possible without violating its maximum capacity.

If we indicate with $m$ the total number of items available and we label each item with a number $j = 1, 2, \ldots, m$, we can represent the selection of the items as a binary vector $x$.

```
Input: Spike train $T_e$, probabilities $P^j_e$, mutation probability $P^{m}_j$, number of ESNPS $H$ and the current best solution $x = (x_1, x_2, \ldots, x_m)$ of length $m$
1: Rearrange $T_e$ as matrix $P_{R_0}$, initialise $gen = 0$ and $P_{m1} = 0$ and then calculate $G_{ij}(0)$ and $D_{P,2}(0)$
2: while ($P_{m1} \leq 1$) do
3: $gen = gen + 1$
4: $i = 1$
5: while ($i \leq H$) do
6: $j = 1$
7: while ($j \leq m$) do
8: if ($rand() < P^{j}_i$) then
9: $k_1, k_2 = ceil(rand * H)$, $k_1 \neq k_2 \neq i$ and correct $x_{k_1}$ and $x_{k_2}$ if violate the constraint
10: if ($f(x^{k_1}) > f(x^{k_2})$) then
11: $x_j = x^{k_1}$
12: else
13: $x_j = x^{k_2}$
14: end if
15: (Adaptive Learning Rate)
16: if ($x_j = 1$) then
17: $P^{j}_i = 0.5 + 0.5P^{j}_i$
18: else
19: $P^{j}_i = 0.5P^{j}_i$
20: end if
21: else
22: $P^{j}_i = 0.5 + 0.5P^{j}_i$
23: end if
24: end if
25: end if
26: end while
27: $j = j + 1$
28: end while
29: $i = i + 1$
30: end while
31: (Adaptive Mutation)
32: calculate $G_{ij}(gen), D_{P,2}(gen)$ and $R_{bestfit}$
33: if ($G_{ij}(gen) > G_{ij}(gen - 1)$) then
34: $P_{m1} = 0$
35: else
36: $P_{m1} = P_{m1} + \frac{1}{\max}$
37: end if
38: $P_{m2} = P_{m2} + \frac{1}{\max}$
39: if ($rand_1() < P_{m1}$ AND $rand_2() > P_{m2}$) then
40: $i = 1$
41: while ($i \leq H$) do
42: $j = 1$
43: while ($j \leq m$) do
44: if ($rand_1() < P^{m}_j$) then
45: $P^{j}_i = rand()$
46: end if
47: $j = j + 1$
48: end while
49: end if
50: $i = i + 1$
51: end while
52: end if
53: end while
Output: Rule probability matrix $P_R$
```

Figure 4. The Dynamic Guider algorithm of AOSNPS

The 0/1 knapsack problem consists of the max-
imisation of the function
\[ f(x) = \sum_{j=1}^{m} p_j x_j \]
subject to
\[ \sum_{j=1}^{m} \omega_j x_j \leq C \]
where \( p_j \) and \( \omega_j \) are price and weight of the \( j^{th} \) item, respectively and \( C \) is the capacity of the knapsack. The parameters of the problem have been set in following way. The weights \( \omega_j \) have been sampled from the interval \([1, \Omega]\), with \( \Omega = 50 \), \( p_j = \omega_j + \frac{1}{2} \Omega \) and the average knapsack capacity \( C \) is applied:
\[ C = \frac{1}{2} \sum_{j=1}^{m} \omega_j. \]

In order to validate the proposed AOSNPS, we tested it against the following algorithms designed/used in the literature
- Genetic Quantum algorithm (GQA)\(^{39}\)
- Novel Quantum Evolutionary algorithm (NQEA)\(^{33}\)
- Optimization Spiking Neural P System (OSNPS)\(^{78}\)

For each algorithm we used the recommended parameter setting used in the original paper. The proposed AOSNPS has been run with \( H = 50 \) ESNPS. Regarding the evolutionary Guider algorithm, the learning probability \( p_j \) \((j = 1, \ldots, m)\) uses the same value used in OSNPS (a random number in the range \([0.05, 0.2])\), the mutation probabilities \( p_j^m = 0.01 \) \((j = 1, \ldots, m)\) and \( N_{max} = 500 \).

The 0/1 knapsack problem has been run in different scenarios (problem instances), i.e. \( m = 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000 \). Each algorithm in this paper has been run for 30 independent runs. When the total weight of all selected items exceeds the capacity \( C \) (constraints violation), we implemented the random chromosome repair technique suggested in Ref.\(^{29,30,75}\).

Table\(^{1}\) displays the numerical results in terms of mean value \( \mu \) ± standard deviation \( \sigma \) of the knapsack cost \( f(x) \). The best results for each problem instance are highlighted in bold.

Furthermore, the statistical significance of the results has been enhanced by the application of the Wilcoxon rank sum test, see\(^{69}\). A \("+"\) indicates that AOSNPS significantly outperforms its competitor, a \("-\) indicates that the competitor significantly outperforms AOSNPS, and a \("=\) indicates that there is no significant difference in performance.

Numerical results in Table\(^{1}\) show that the proposed AOSNPS is very efficient to address the 0/1 knapsack problem since it achieves the best results in nine cases out of the ten considered. In only one scenario (with the smallest number of items \( m = 1000 \)) NQEA achieves slightly better results than AOSNPS. The effectiveness of the novel Dynamic Guider algorithm appears from the direct comparison against OSNPS: AOSNPS systematically outperforms OSNPS.

In order to further strengthen the statistical analysis of the presented results, we performed the Holm-Bonferroni\(^{31}\) procedure for the eight algorithms (\( N_A = 4 \)) and ten problem instances (\( N_p = 10 \)). The rank \( R_k \) for \( k = 1, \ldots, N_A \) by assigning, for each problem instance has been calculated. For each problem instance, a score \( N_A \) is assigned to the best algorithm, a score \( N_A - 1 \) to the second best, \( \ldots \), 1 to the worst algorithm. The ranks \( R_k \) are the scores averaged over all the problem instances. Let us indicate with \( R_{l0} \) the ranking of AOSNPS. For the remaining \( N_A - 1 \) algorithm the score \( z_k \) is calculated as the values \( z_k \) have been calculated as:
\[ z_k = \frac{R_k - R_{l0}}{\sqrt{\frac{N_A(N_A+1)}{6N_p}}}. \]

By means of the \( z_k \) values, the corresponding cumulative normal distribution values \( p_k \) have been calculated, see\(^{19}\):
\[ p_k = \frac{2}{\sqrt{\pi}} \int_{-\infty}^{z_k} e^{-t^2} dt. \]

These \( p_k \) values have then been compared with the corresponding \( \delta/k \) where \( \delta \) is the level of confidence, set to 0.05 in this case.

These \( p_k \) values have then been compared with the corresponding \( \delta/k \) where \( \delta \) is the level of confidence, set to \( \delta = 0.05 \) in this case. Table\(^{2}\) displays the ranks, \( z_k \) values, \( p_k \) values, and corresponding \( \delta/k \) obtained. Moreover, it is indicated whether the null-hypothesis (that the two algorithms have indistinguishable performances) is "Rejected", i.e. the algorithms have statistically different performance, or
“Accepted” if the distribution of values can be considered the same (there is no outperformance).

The Holm-Bonferroni procedure in Table 2 shows that AOSNPS achieves the best performance over all the algorithms taken into account. It can be observed that AOSNPS achieves the best ranking and significantly outperforms GQA and OSNPS.

Table 2. Holm-Bonferroni Procedure with AOSNPS as reference (Rank 3.9000e+00)

<table>
<thead>
<tr>
<th>mil</th>
<th>GQA</th>
<th>NQEA</th>
<th>OSNPS</th>
<th>AOSNPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>µ</td>
<td>σ</td>
<td>µ</td>
<td>σ</td>
<td>µ</td>
</tr>
<tr>
<td>1000</td>
<td>2634.0617</td>
<td>162.941</td>
<td>2927.375</td>
<td>131.036</td>
</tr>
<tr>
<td>3000</td>
<td>76259.435</td>
<td>766.285</td>
<td>83866.620</td>
<td>302.899</td>
</tr>
<tr>
<td>4000</td>
<td>103801.413</td>
<td>1128.401</td>
<td>113991.532</td>
<td>666.620</td>
</tr>
<tr>
<td>5000</td>
<td>131224.181</td>
<td>1312.772</td>
<td>146075.755</td>
<td>916.934</td>
</tr>
<tr>
<td>6000</td>
<td>157119.378</td>
<td>1571.194</td>
<td>169516.765</td>
<td>1162.772</td>
</tr>
<tr>
<td>7000</td>
<td>182669.798</td>
<td>191659.447</td>
<td>208561.466</td>
<td>1162.772</td>
</tr>
<tr>
<td>8000</td>
<td>208561.778</td>
<td>233945.639</td>
<td>254551.819</td>
<td>1162.772</td>
</tr>
<tr>
<td>9000</td>
<td>233945.831</td>
<td>276903.178</td>
<td>29273.756</td>
<td>254551.819</td>
</tr>
</tbody>
</table>

4.1 Comparative Analysis of AOSNPS and OSNPS

In this subsection, we highlight the effectiveness of the new Dynamic Guider algorithm by comparing the behaviour of OSNPS and AOSNPS. The analysis involves the following five indicators:

(1) $G_{bf}$: performance trend. This is the objective function value of the current best solution at each generation $gen$.

(2) $D_{hbw}$: Hamming distance between the best and worst binary individuals in a population.

$$D_{hbw} = \frac{1}{m} \sum_{i=1}^{m} (x_{bi} \oplus x_{wi})$$

where, $x_{bi}$ and $x_{wi}$ are the $i$th bits in the best and worst binary solutions, respectively; $m$ is the number of bits in a binary solution; the symbol $\oplus$ represents the OR operator.

(3) $D_{hm}$: average Hamming distance of all binary individuals in a population.

$$D_{hm} = \frac{2}{n(n-1)} \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{1}{m} \sum_{k=1}^{m} (x_{ik} \oplus x_{jk})$$

where, $x_{ik}$ and $x_{jk}$ are the $k$th bits in the $i$th and $j$th binary solutions, respectively; $m$ is the number of bits in a binary solution; $n$ is the number of individuals in a population; the symbol $\oplus$ represents OR operator.

(4) $DP_{bw}$: distance between the best and worst probability column vectors in $P_R$ corresponding to the best and worst fitness values in a population, respectively.

$$DP_{bw} = \frac{1}{m} \sum_{k=1}^{m} \left| p_{bk}^{1} - p_{wk}^{1} \right|$$

where $p_{bk}^{1}$ and $p_{wk}^{1}$ are the probabilities associated with the best and worst.

(5) $DP_{a}$: average probability distance of all probability individuals as shown in Subsection 3.2

Figure 5. Performance trend $G_{bf}$ of OSNPS and AOSNPS

Figures 3 - 7 depicts the trends of these five metrics for $m = 1000$ items for OSNPS and AOSNPS, respectively. The algorithms have been run with the same starting random initial probability
matrices $P_{50 \times 1000}$. Both OSNPS and AOSNPS have been run 8000 generations respectively. Figure 5 shows that AOSNPS outperforms OSNPS throughout the entire run. Figures 6 and 7 show that AOSNPS exhibits better transient convergence: the diversity metrics $D_{hbw}$ and $D_{hm}$ of AOSNPS drop more rapidly than that of OSNPS in the early stages of the run. However, subsequently AOSNPS achieves and maintains a better diversity than OSNPS. This is the result of the joint contributions of the adaptive learning rate (the rapid decline of diversity in early stage is mainly due to the improved learning efficiency using $\Delta_{adaptive}$) and the adaptive mutation (better diversity obtained in the middle and late stages is due to the frequent triggering of mutation rule).

Figure 6. Plot of Hamming distance between the best and worst binary individuals $D_{hbw}$: OSNPS vs AOSNPS

Numerical results of $DP_{bw}$ and $DP_{a}$ in Fig 8-9 highlight the better ability of AOSNPS to maintain the diversity of probability individuals comparing with OSNPS. This is due to the contribution of the adaptive mutation. During the entire run of AOSNPS (8000 generations), the adaptive mutation was triggered 1971 times.

Figure 8. Plot of the distance between the best and worst probability $DP_{bw}$: OSNPS vs AOSNPS

Figure 9. Plot of the average probability distance $DP_{a}$: OSNPS vs AOSNPS

Fig 10 shows the histogram of the mutation
triggering during the run. At the beginning of the algorithm execution, the mutation was not enabled because the optimal solution improved quickly. In the middle and at the late stages of the run, the mutation is often enabled to enhance upon a current best solution with a high performance.

In order to better display the behavior of adaptive mutation, Fig. 11 displays the interval generations, i.e., the number of generations \( \text{gen} \) between two consecutive triggered mutations. When the interval generation is one, then the mutation has occurred for two consecutive generations. At this time, the global optimal solution may have not been improved for many generations (\( P_{m1} \) is large), and the difference between the populations is small (\( P_{m2} \) is small), which could correspond to a convergence to a suboptimal solution. In this situation, mutations are needed to explore possible new promising search directions. If the interval generation of mutation is large, either AOSNPS is likely to have recently detected a new global optimal solution (\( P_{m1} \) is reset to zero) or the difference between the populations is large (\( P_{m2} \) is large).

Figure 10. The histogram of the mutation triggering

Figure 11. The interval generations between two adjacent mutations triggered

Figure 12. IEEE 39 bus power system

5. AOSNPS for Power System Fault Diagnosis

Power system fault diagnosis is the identification of location and causes of a fault on a power system. In this subsection, AOSNPS is applied to solve this fault diagnosis problem. We considered the power system represented in Fig. 12 see Ref. [63]. The power system contains 39 buses, 45 lines and 99 Circuit breakers. 39 buses and 45 lines are \( B_1, \ldots, B_{39} \), and, \( L_{1-7}, \ldots, L_{29-39} \), 99 Circuit breakers are \( CB_{(1)-7}, \ldots, CB_{(39)-29} \), where, \( CB_{(1)-7} \) represents the breaker such that \( L_{1-7} \) is near the \( B_1 \)
side. For instance, line $L_{3-4}$ has three types of protective relays including main protective relay $L_{(3-4)m}$, first backup protective relay $L_{(3-4)p}$ and second backup protective relay $L_{(3-4)r}, L_{(3-4)s}, L_{(3-5)s}$ and $L_{(5-8)s}$. The operational rules of the protective relays of buses and lines are described in Ref.[33] as follows.

1) Protective relays of buses. If the main protective relays of a bus operate, all breakers connected to the bus are tripped.

2) Protective relays of lines. If the main protective relays of a line operate, all breakers connected to the line are tripped. Likewise, when the main protective relays of a bus fail to operate, the first backup protective relays operate to trip all breaker connected to the line. When adjacent regions of a line fail and their main protective relay and first backup relay fail to operate, the second backup relay of a line operates.

Six fault cases in the IEEE 39 bus system and IEEE 118 bus system have been considered: case 1 is a single fault, case 2 is multiple faults, case 3 and case 4 are multiple faults with incompleteness and uncertainty in the IEEE 39 node electric power system. Case 5 is multiple faults and case 6 is multiple faults with incompleteness and uncertainty in the IEEE 118 node electric power system[5]. The network data for the four cases are listed in Table 3.

In order to identify a fault, the engineer is usually informed about the real status of protective relays and circuit breakers. We represent this piece as information as two binary vectors $r$ and $c$ for relays and circuit breakers, respectively.

\[
\begin{align*}
\mathbf{r} &= (r_1, r_2, \ldots, r_{Nr}) \\
\mathbf{c} &= (c_1, c_2, \ldots, c_{Nc})
\end{align*}
\]

where for every value of $i$, 1) $r_i = 1$ and $r_i = 0$ represent the operation and non-operation of the protective relay $i$; 2) $c_i = 1$ and $c_i = 0$ represent the operation (the circuit breakers trips) and non-operation of the circuit breaker. Let us now define the binary vector $s$,

\[
\mathbf{s} = (s_1, s_2, \ldots, s_n)
\]

where $s_i$ represents the status of section: $s_i = 1$ and $s_i = 0$ represent the fault status and normal status of section $i$, respectively.

On the basis of these pieces of information, the expected values of the protective relays and circuit breakers

\[
\begin{align*}
\mathbf{re} &= (re_1, re_2, \ldots, re_{Nr}) \\
\mathbf{ce} &= (ce_1, ce_2, \ldots, ce_{Nc})
\end{align*}
\]

are represented as functions of $r$ and $s$ and computed as shown in Ref. [65]. We used AOSNPS to optimise the following objective function[62]

\[
e(s) = \sum_{i=1}^{Nr} |r_i - re_i(s)| + \sum_{j=1}^{Nc} |c_j - ce_j(s, r)|
\]

Table 4 displays the results of the fault diagnosis estimation obtained by AOSNPS and compare them with those obtained by the ad-hoc expert based methods presented in Ref. [61] and [72]. It can be observed that for case 1 the three methods reach the same results. Regarding case 2, the estimation result of AOSNPS is the same as [72] but is different to [72]. In case 3, case 4, case 5 and case 6, we may observe that the results of AOSNPS are different from those in [61] and [72]. However, our results agree with those achieved by a more recent study that considers only complex scenarios, see Ref.[67].

6. Conclusions

This paper proposes a novel Adaptive Spiking Neural P System, indicated as AOSNPS, for solving the 0/1 knapsack problem. The proposed method uses multiple neural P systems, each of them generating a candidate solution and a Dynamic Guider algorithm that supervises the search. The proposed scheme includes two novel adaptive mechanisms within the Dynamic Guider algorithm: an adaptive learning rate and an adaptive mutation based on the estimation of the diversity of the sampling rules.

Numerical results demonstrate that thanks to their adaptive learning rate and adaptive mutation, the proposed AOSNPS overcomes the diversity loss limitation of its predecessor OSNPS. The novel adaptive mechanism enhances upon the behaviour of the Dynamic Guider algorithm: the search operators, through the probabilities of the spiking neural P system, react to the temporary diversity loss of the population by suggesting fresh search directions. The proposed AOSNPS outperforms two popular metaheuristics and OSNPS on the 0/1 knapsack problem in all the scenarios considered. Furthermore, the application of AOSNPS to a power distribution problem shows that AOSNPS behaves like...
Table 3. Status information about four cases

<table>
<thead>
<tr>
<th>case</th>
<th>protective relays</th>
<th>breakers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$L_{(18)-(19)m}$ operated, $L_{(18)-(19)p}$ operated</td>
<td>$CB_{(18)-(19)}$ operated, $CB_{18-(19)}$ operated</td>
</tr>
<tr>
<td>2</td>
<td>$L_{(18)-(19)m}$ operated, $L_{(18)-(19)p}$ operated</td>
<td>$CB_{(18)-(19)}$ operated, $CB_{18-(19)}$ operated, $CB_{17-(19)}$ operated, $CB_{19-(23)}$ operated, $CB_{18-(19)}$ refused to operate</td>
</tr>
<tr>
<td>3</td>
<td>$B_{19m}$ operated, $L_{(17)-(19)}$ operated</td>
<td>$CB_{(18)-(19)}$ operated, $CB_{18-(19)}$ operated, $CB_{17-(19)}$ operated, $CB_{19-(23)}$ operated, $CB_{17-(19)}$ refused to operate</td>
</tr>
<tr>
<td>4</td>
<td>$L_{(11)-(12)m}$ operated, $L_{(11)-(12)p}$ operated</td>
<td>$CB_{(11)-(12)}$ operated, $CB_{12-(13)}$ operated, $CB_{17-(19)}$ operated, $CB_{17-(19)}$ refused to operate</td>
</tr>
<tr>
<td>5</td>
<td>$B_{83m}$ operated, $L_{(19)-(20)m}$ operated</td>
<td>$CB_{(9)-(10)}$ operated, $CB_{7-(8)}$ operated, $CB_{11-(12)}$ operated, $CB_{19-(20)}$ operated, $CB_{21-(22)}$ operated, $CB_{9-(10)}$ refused to operate</td>
</tr>
<tr>
<td>6</td>
<td>$B_{85m}$ operated, $L_{(3)-(4)m}$ operated</td>
<td>$CB_{(3)-(4)}$ operated, $CB_{3-(4)}$ operated, $CB_{5-(6)}$ operated, $CB_{7-(10)}$ operated, $CB_{13-(14)}$ operated, $CB_{15-(16)}$ operated, $CB_{15-(16)}$ refused to operate, $CB_{12-(22)}$ refused to operate</td>
</tr>
</tbody>
</table>

Table 4. Comparisons between AOSNPS and two other methods for power grid fault detection

<table>
<thead>
<tr>
<th>case</th>
<th>candidate sections</th>
<th>The results in Ref.[61]</th>
<th>The results in Ref.[72]</th>
<th>AOSNPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$L_{18-19}$</td>
<td>$L_{18-19}$</td>
<td>$L_{18-19}$</td>
<td>$L_{18-19}$</td>
</tr>
<tr>
<td>2</td>
<td>$L_{17-19}, L_{18-19}, L_{19-23}$</td>
<td>$L_{17-19}, L_{18-19}$</td>
<td>$L_{18-19}$</td>
<td>$L_{18-19}$</td>
</tr>
<tr>
<td>3</td>
<td>$L_{17-19}, L_{18-19}, L_{19-23}$</td>
<td>$B_{19}, L_{17-19}, L_{18-19}$</td>
<td>$B_{19}, L_{17-19}, L_{18-19}$</td>
<td>$B_{19}, L_{18-19}$</td>
</tr>
<tr>
<td>4</td>
<td>$B_{12}, L_{11-12}, L_{12-13}, L_{17-19}$</td>
<td>$B_{12}, L_{12-13}, L_{17-19}$</td>
<td>$B_{12}, L_{11-12}, L_{17-19}$</td>
<td>$L_{11-12}, L_{17-19}$</td>
</tr>
<tr>
<td>5</td>
<td>$B_{83}, L_{9-10}, L_{19-20}, L_{21-22}$</td>
<td>$B_{83}, L_{9-10}, L_{19-20}, L_{21-22}$</td>
<td>$B_{83}, L_{19-20}, L_{21-22}$</td>
<td>$B_{83}, L_{19-20}, L_{21-22}$</td>
</tr>
<tr>
<td>6</td>
<td>$B_{85}, B_{86}, L_{3-4}, L_{1-2}, L_{7-16}$</td>
<td>$B_{85}, B_{86}, L_{3-4}, L_{7-16}$</td>
<td>$B_{85}, B_{86}, L_{3-4}, L_{7-16}$</td>
<td>$B_{85}, L_{3-4}$</td>
</tr>
</tbody>
</table>

an artificial intelligence element able to correctly locate electrical faults.

Acknowledgments
This work was supported the National Natural
Bibliography

28. S. Ghosh-Dastidar, H. Adeli and N. Dadmehr, Prin-
principal component analysis-enhanced cosine radial basis function neural network for robust epilepsy and seizure detection, **IEEE Transactions on Biomedical Engineering** **55**(2) (2008) 512–518.


52. A. Panakkat and H. Adeli, Recurrent neural network for approximate earthquake time and location prediction using multiple seismicity indicators, **Computer-Aided Civil and Infrastructure Engineering** **24**(4) (2009) 280–292.


55. W. Pitts and W. McCulloch, A logical calculus of the ideas immanent in nervous activity, **Bull. Math. Biophys** **5**(1943) 115–133.


58. T. Song, A. Rodríguez-Patón, P. Zheng and X. Zeng, Spiking neural P systems with colored spikes, **IEEE Transactions on Cognitive and Development Systems**


