An Adaptive Memetic P System to Solve the 0/1 Knapsack Problem

Jianping Dong, Haina Rong
School of Electrical Engineering
Southwest Jiaotong University
Chengdu, China
1969140573@qq.com; ronghaina@126.com

Ferrante Neri
COL Lab, School of Computer Science
University of Nottingham
Nottingham, United Kingdom
ferrante.neri@nottingham.ac.uk

Qiang Yang, Ming Zhu
School of Control Engineering
Chengdu University of information Technology
Chengdu, China
yuxia2008@126.com, zhuming@cuit.edu.cn

Gexiang Zhang
College of Information Science and Technology
Chengdu University of Technology
Chengdu, China
zhgxdylan@126.com

Abstract—Memetic Algorithms are traditionally composed of an evolutionary framework and one or more local search elements. However, modern generation Memetic Algorithms do not necessarily follow a pre-established scheme and are hybrid structures of various types. By following these modern trends, the present paper proposes an original and unconventional adaptive memetic structure generated by the hybridisation of a set of theoretical computational models, namely P Systems, and an evolutionary algorithm employing adaptation rules and moving operators inspired by Evolution Strategies. The resulting memetic algorithm, namely Adaptive Optimisation Spiking Neural P System (AOSNPS), is a tailored algorithm to solve optimisation problems with binary encoding.

More specifically AOSNPS is composed of a family of parallel spiking neural P systems, each of them generating a binary vector representing a candidate solution on the basis of internal probability parameters and an adaptive Evolutionary Guider Algorithm that evolves the probabilities encoded in each P system. Numerical result shows that the proposed approach is effective to solve the 0/1 knapsack problem and outperforms various algorithms proposed in the literature to solve the same class of problems.

Index Terms—Memetic Algorithms, P Systems, Membrane Computing, Evolutionary Algorithms, Knapsack Problem

I. INTRODUCTION

In their early implementations, Memetic Algorithms (MAs) were identified with an evolutionary algorithm employing a local search [1], [2]. Subsequently, MAs no longer referred to a specific algorithmic structure but progressively became a strategy about the design of algorithms by using multiple and diverse search operators, see [3].

This fact has been emphasised in the review paper [4] which categorises MAs into three major groups

- Simple Hybrids: this category includes hybrid algorithms generated by two or more algorithms (usually one global and at least one local) combined together in a synergistic manner, see [1]. Some hybridisations belonging to this category can be very efficient to address specific domains, see e.g. [5]–[8].
- Adaptive Hybrids: this category includes hybrid algorithms where multiple local search algorithms are coordinated by a supervising adaptive mechanism e.g. performance-based like hyperheuristics [9] and meta-Lamarckian learning [10], diversity-based [11], [12] or self-adaptation, see [13].
- (Future) Memetic Automation: this category is a visionary idea that sees memetic algorithms fully self-generated by machines as a combination of “agents”, see e.g. [14], [15]. Although this design approach is still under investigation, some interesting domain-specific frameworks [16], [17] and prototypes [18] have been proposed.

Thus, simple and adaptive hybrids tend to have a prearranged structure composed of an evolutionary framework and one or more local searches (possibly adaptively activated) that enhance upon the performance of one solution [19]. Conversely, modern MAs are thought as algorithms resulting from any hybridisation as long as they are capable to efficiently address a class of problems.

By following this consideration, this paper proposes an MA based on an unconventional hybridisation to solve binary problems and focuses on the 0/1 knapsack problem. More specifically, the proposed algorithm makes use of an adaptive evolutionary framework which includes an adaptive variation operator inspired by Evolutionary Strategy (ES) and another mutation operator that re-samples the points on the basis of the diversity and historical data. This evolutionary framework, namely Evolutionary Guider Algorithm, does not vary the solutions of the knapsack problem but varies the probabilities of the neurons of a family of Spiking Neural P Systems (SNPSs) [20], [21]. A SNPS is a general purpose computational structure belonging to the field of Membrane Computing, see [22]–[24]. In the context of the proposed MA, the SNPSs arranged in a family are used as number generators.
to generate strings of binary numbers (the candidate solutions of the 0/1 knapsack problem). Hence, the Evolutionary Guider Algorithm modifies and detects the parameters that control the generation of the candidate solutions.

The resulting algorithm namely Adaptive Optimisation Spiking Neural P System (AOSNPS) enhances a previous algorithm based on a family of P systems proposed in [25].

The remainder of this article is organised in the following way. Section II briefly introduces the 0/1 knapsack problem, provides the basic definition of P systems (Subsections II-A) and describes the algorithm in [25] (Subsections II-B and II-C). Section III describes the proposed AOSNPS. Section IV displays the numerical results of the proposed approach in different scenarios and against multiple algorithms used to solve this problem. Finally, Section V provides the conclusions of this study.

II. BACKGROUND: 0/1 KNAPSACK PROBLEM AND OPTIMISATION BY P SYSTEMS

In this paper we aim at solving the 0/1 Knapsack Problem which is formulated as follows. 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 \( j = 1 \ldots m \), we may represent the selection of the items as a vector

\[
x = (x_1, x_2, \ldots, x_m)
\]

of binary numbers.

Then, the 0/1 Knapsack problem can be expressed as the maximisation 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 the following subsections, we briefly summarise the relevant theory about P systems and present the Optimisation Spiking Neural P System (OSNPS) introduced in [25] to solve the 0/1 knapsack problem.

A. Spiking Neural P System

The SNPS [20], [26] is an automaton represented by the tuple

\[
\Pi = (O, \sigma_1, \ldots, \sigma_m, \text{syn}, \sigma_{\text{out}})
\]

where

1) \( O = \{a\} \) is the singleton alphabet, \( a \) is called spike
2) \( \sigma_1, \ldots, \sigma_m \) are neurons, where each neuron \( \sigma_i \) is a pair

\[
\sigma_i = (n_i, R_i)
\]

with

(a) \( n_i \geq 0 \) initial number of spikes contained in \( \sigma_i \), e.g. if the neuron \( \sigma_i \) has an initial number of spikes \( n_i = 5 \), it contains the work \( aaaaa = a^5 \)
(b) \( R_i \) set of the two following rules:

(i) \textbf{spiking rule}: \( E/a^c \rightarrow a; d \) where \( E \) is a regular expression over \( O \), and \( c \geq 1, d \geq 0 \). The spiking rule means that a neuron containing a word \( a^c \) may lose \( c \leq n_i \) spikes after a delay of \( d \) steps

(ii) \textbf{forgetting rule}: \( a^s \rightarrow \lambda \), for some \( s \geq 1 \), with the restriction that for each rule \( E/a^c \rightarrow a; d \) of type (i) from \( R_i \), we have \( a^s \notin L(E) \). The forgetting rule means that a neuron can lose all its \( s \) spikes as long as \( a^s \) does not belong to the language \( L(E) \) identified by the regular expression \( E \).

3) the connections between neurons are described by the synapses \( \text{syn} \) where \( \text{syn} \subseteq \{\sigma_1, \ldots, \sigma_m\} \times \{\sigma_1, \ldots, \sigma_m\} \)

with \( (\sigma_i, \sigma_j) \notin \text{syn} \) for \( i = 1, \ldots, m \). The synapses govern the propagation of the spikes, if the neuron \( \sigma_i \) spikes and \( (\sigma_i, \sigma_j) \in \text{syn} \) then the same spiking action is repeated on the neuron \( \sigma_j \)
4) the output neuron \( \sigma_{\text{out}} \in \{\sigma_1, \ldots, \sigma_m\} \).

The distinguishing feature of SNPS is that an input causes a nondeterministic spike train in the output, see [27]. If the output neuron spikes, then we code this behaviour as a 1 and otherwise we code it as a 0. Hence, the spike train can be represented by a sequence of ones and zeros. Details about the architecture of SNPS are described and discussed in detail in [20], [25], [28].

B. Extended Spiking Neural P System

In [25], in order to exploit the potentials of SNPSs for addressing an optimisation problem, the SNPS has been extended and slightly modified. The resulting implementation of P system, namely Extended Spiking Neural P System (ESNPS) is a tuple of the type

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

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 ($n_i = 1, \forall i$)
(b) $R_i = \{r^1_i, r^2_i\}$ is a set of rules, spiking $r^1_i : a \rightarrow a$ and forgetting $r^2_i : a \rightarrow \lambda$, respectively.
(c) $P_i = \{p^1_i, p^2_i\}$ is a finite set of probabilities, where $p^1_i$ and $p^2_i$ are the selection probabilities of rules $r^1_i$ and $r^2_i$, respectively, and satisfy $p^1_i + p^2_i = 1$.

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) $s_{in} = \{(\sigma_i, \sigma_j) | (1 \leq i \leq m + 1 \text{ AND } j = m + 2 \text{ OR } (i = m + 2 \text{ AND } j = m + 1)\}$.
5) $I_0 = \{1, 2, \ldots, m\}$ is a finite set of output neurons, i.e., the output is a spike train formed by concatenating the outputs of $\sigma_1, \sigma_2, \ldots, \sigma_m$.

The structure of an ESNPS is shown in Fig. 1.

![Fig. 1. Logical and functioning scheme of ESNPS](image)

C. Optimisation 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.

![Fig. 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 $i$. The input of the Guider Algorithm is a spike train $T_s$, 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 $i$. The output of the Guider Algorithm is the rule probability matrix $P_R = [p^1_{ij}]_{H \times m}$, 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.

Algorithm 1 provides the details of the Guider Algorithm used in OSNPS [25]. 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 $p^0_{ij}$, and compared on the basis of a fitness (objective function) $f$ that measures the value/profit of the knapsack. At each objective function call, in order to handle the case when the total weight of all selected items exceeds the capacity $C$, we implemented the random chromosome repair technique suggested in [29]–[31].

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 Algorithm 1. More details on OSNPS can be found in [25].

III. ADAPTIVE OPTIMISATION SPIKING NEURAL P SYSTEM

The proposed AOSNPS employs the same structure of OSNPS depicted in Fig. 2 but hybridises the theoretical computer science logic of ESNPSs with that of Evolutionary Algorithms (EAs) [32] to manipulate the probability matrix $P_R$. More specifically, the Guider Algorithm is an EA embedding two elements inspired by Evolution Strategies (ES) [33], [34] within it: 1) an adaptive learning rate $\Delta$; 2) an adaptive mutation. The novel mechanisms are described in Subsections III-A and III-B, respectively. The new Guider Algorithm embedding these two elements is outlined in Subsection III-C.

A. Adaptive Learning Rate

The learning rate $\Delta$ is the step size of probability adjustment for the elements of the probability matrix $P_R$:

$$p^i_{ij} = p^i_{ij} + \Delta$$
$$p^i_{ij} = p^i_{ij} - \Delta$$

with $1 \leq i \leq H$ and $1 \leq j \leq m$. In [25], $\Delta$ is a random number between 0.005 and 0.02 set at the beginning of the optimisation 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 \( p_{ij} \) of ESNSP presented in Section II-B.

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

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

\[
\Delta_{ij}^a = \frac{P_b - p_{ij}^1}{2}
\]

\( \Delta_{ij} \) is designed to take the middle point of the distance between the current probability \( p_{ij} \) and the ideal probability. The ideal probability \( P_b = \{0,1\} \) is the lower or upper bound of the probability of \( p_{ij} \) (the pedex 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 OSNSP, 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 by following the learning needs. If the distance between the current probability \( p_{ij} \) and the ideal probability \( P_b \) is big, the learning rate \( \Delta_{ij} \) is big. On the other hand, if the distance between the current probability \( p_{ij} \) 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 \( p_{ij} = 0.1 \) the proposed adaptive learning rate \( \Delta_{ij}^a \) enables to reach 0.9 < \( p_{ij} \) < 1 after four steps whereas with a constant \( \Delta \) over 40 steps are needed.

- The probability of \( p_{ij} \) does not overflow. In OSNSP, it may result \( p_{ij}^1 > 0 \) or \( p_{ij}^1 < 0 \). In this case, an extra mechanism is required. The use of the proposed adaptive learning rate ensures that the probability overflow never occurs.

B. Adaptive Mutation

The proposed AOSNPS makes also use of an adaptive mutation strategy. Two dynamic parameters \( P_{m1} \) and \( P_{m2} \) are defined to characterise and monitor the evolutionary state of AOSNPS. The parameter \( P_{m1} \) is the first mutation probability which varies between 0 and 1 and is defined as follows:

\[
\text{if } G_{bf}(gen) > G_{bf}(gen - 1) \text{ then } P_{m1} = 0
\]

\[
\text{end if}
\]

\[
\text{if } G_{bf}(gen) = G_{bf}(gen - 1) \text{ then } P_{m1} = P_{m1} + \frac{1}{N_{max}}
\]

\[
\text{end if}
\]

---

**Algorithm 1** The Guider Algorithm of OSNSP

**Require:** Spike train \( T_s \), number of ESNSP \( H \) and the current best solution \( x = (x_1, x_2, \ldots, x_m) \) of length \( m \)

1. Rearrange \( T_s \) as matrix \( P_R \)
2. \( i = 1 \)
3. while \( (i \leq H) \) do
   4. \( j = 1 \)
   5. while \( (j \leq m) \) do
     6. if \( \text{(rand()} < p_{ij}^3) \) then
       7. \( k_1, k_2 = \text{ceil} \left( \text{rand()} \times H \right), k_1 \neq k_2 \neq i \) and correct \( x_{k_1} \) and \( x_{k_2} \) if violate the constraint
     8. else
       \( x_j = x_{k_i} \)
     9. end if
   10. \( x_j = x_{k_2} \)
   11. if \( (x_j = 1) \) then
     12. \( p_{ij}^1 = p_{ij}^1 + \Delta \)
   13. \( \text{else} \)
     14. \( p_{ij}^1 = p_{ij}^1 - \Delta \)
   15. end if
   16. \( x_j = x_{k_1} \)
   17. if \( (x_j = 1) \) then
     18. \( p_{ij}^1 = p_{ij}^1 + \Delta \)
   19. \( \text{else} \)
     20. \( p_{ij}^1 = p_{ij}^1 - \Delta \)
   21. end if
   22. \( x_j = x_{k_1} \)
   23. if \( (p_{ij}^1 > 0) \) then
     24. \( p_{ij}^1 = p_{ij}^1 - \Delta \)
   25. \( \text{else} \)
     26. \( p_{ij}^1 = p_{ij}^1 + \Delta \)
   27. end if
   28. \( j = j + 1 \)
   29. \( i = i + 1 \)
30. end while
31. end if
32. Ensure: Rule probability matrix \( P_R \)
where \( G_{bf}(gen) \) 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 \( N_{\text{max}} \) determines a stopping criterion; if the global best fitness does not improve for consecutive \( N_{\text{max}} \) generations the algorithm stops.

The parameter \( P_{m2} \) is the second mutation probability and is defined as follows:

\[
P_{m2} = \frac{DP_a(\text{gen})}{DP_a(0)}
\]

where \( DP_a(\text{gen}) \) is an aggregated metric representing the diversity (of probabilities) at the current generation \( \text{gen} \), see [11], [12]. This value is calculated as:

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

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

The triggering rule for adaptive mutation is defined as

\[
rand_1() < P_{m1} \quad \text{AND} \quad rand_2() > P_{m2}
\]

where \( rand_1() \) and \( rand_2() \) are two random numbers in the range \([0, 1]\).

The logic of the mutation probabilities 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 \( (P_{m1} = 0) \). 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 \( (P_{m1} = P_{m1} + \frac{1}{N_{\text{max}}}) \).
- 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 \((G_{bf} \) 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 \( (P_{m1} \) is large, \( P_{m2} \) is small).

When triggered the mutation of the matrix \( P_R \) is performed in the following way:

\[
\text{for } i = 1, 2, \ldots, H \text{ with } i \neq R_{\text{bestfit}} \text{ do} \\
\quad \text{for } j = 1, 2, \ldots, m \text{ do} \\
\quad \quad \text{if } rand_3() < P_{m}^{ij} \text{ then} \\
\quad \quad \quad p_{ij}^{\text{new}} = rand() \\
\quad \quad \text{end if} \\
\quad \text{end for} \\
\text{end for}
\]

where \( rand_3() \) and \( rand() \) are two random numbers in the range \([0, 1]\). The mutation probability \( P_{m}^{ij} \) is a parameter sampled in the range \([0, 0.1]\) at the beginning of the Guider Algorithm’s execution. \( R_{\text{bestfit}} \in [1, H] \) 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 to the mutation.

C. The Evolutionary Guider of Adaptive Optimisation Spiking Neural P System

Based on the adaptive learning rate and the adaptive mutation rule, the Guider Algorithm is an EA operating on the learning probabilities to support the generation of successful solution for the 0/1 knapsack 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 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]–[31].

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 and described in Subsection III-A, the re-initialisation of the neuron probability which is triggered only by the satisfaction of the conditions described in Subsection III-B. Hence, we may consider AOSNPS as a co-evolutionary memetic algorithm, see [8], [35], [36], where the candidate solutions of the 0/1 knapsack problem evolve with the probabilities of the neurons in each of the P systems. Each P system can be seen as an local search algorithm (each of them depends on different probabilities). In this unconventional memetic algorithm, the probabilities are directly manipulated by an adaptive EA, the Evolutionary Guider Algorithm, while the fitness refers only to the solutions generated by the P systems. The latter have the role of searching the decision space on the basis of the probabilities determined by the Evolutionary Guider Algorithms.

The pseudocode of the proposed Evolutionary Guider is shown in Algorithm 2.

IV. Numerical Results

In order to validate the proposed AOSNPS, we tested it against the following algorithms designed/used in the literature to solve the 0/1 knapsack problem

- Genetic Quantum Algorithm (GQA) [37]
- Novel Quantum Evolutionary Algorithm (NQEA) [38]
- Optimisation Spiking Neural P System (OSNPS) [25]

For each algorithm we used the recommended parameter setting used in the original paper. The proposed AOSNPS has been run with \( H = 50 \) ESNPS with 1002 neurons each. Regarding the Evolutionary Guider Algorithm, the learning probability \( P_{m}^{ij} (j = 1, \ldots, m) \) uses the same value used in OSNPS (a random number in the range \([0.05, 0.2]\), the mutation probabilities \( P_{m}^{ij} = 0.01 (j = 1, \ldots, m) \) and \( N_{\text{max}} = 500 \).

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
Algorithm 2 The Evolutionary Guider Algorithm of AOSNPS

Require: Spike train $T_s$, probabilities $p_{ij}$, mutation probability $P_m$, number of ESNPS $n$ and the current best solution $\mathbf{x} = (x_1, x_2, \ldots x_m)$ of length $m$
1: Rearrange $T_s$ as matrix $P_R$, initialise $gen = 0$ and $P_{m1} = 0$
2: while ($P_{m1} < 1$) do
3:   $gen = gen + 1$
4:   $i = 1$
5:   while ($i < H$) do
6:      $j = 1$
7:      while ($j \leq m$) do
8:         if ($rand_i < p_{ij}^o$) then
9:            $k_1, k_2 = ceil(rand \cdot H)$, $k_1 \neq k_2 \neq i$ 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:               $p_{ij} = 0.5 + 0.5 p_{ij}^o$
18:            else
19:               $p_{ij} = 0.5 p_{ij}^o$
20:            end if
21:         else
22:            if ($x^{max} = 1$) then
23:               $p_{ij} = 0.5 + 0.5 p_{ij}^o$
24:            else
25:               $p_{ij} = 0.5 p_{ij}^o$
26:            end if
27:         end if
28:      end while
29:      $i = i + 1$
30:   end while
31:   $\{Adaptive\ Mutation\}$
32:   calculate $G_{bf}(gen)$, $DP_{bf}(gen)$ and $R_{bestfit}$
33:   if ($G_{bf}(gen) > G_{bf}(gen - 1)$) then
34:      $P_{m1} = 0$
35:   else
36:      $P_{m1} = P_{m1} + \frac{1}{n_{max}}$
37:   end if
38:   if ($rand1() < P_{m1} \text{ AND } rand2() > P_{m2}$) then
39:      $i = 1$
40:   end while
41:   $\{Adaptive\ Mutation\}$
42:   if ($i \neq R_{bestfit}$) then
43:      $j = 1$
44:      while ($j \leq m$) do
45:         if ($rand1() < p_{ij}^m$) then
46:            $p_{ij} = rand()$
47:         end if
48:      end while
49:   end if
50:   $i = i + 1$
51: end while
52: end if
53: end while

Ensure: Rule probability matrix $P_R$
In order to emphasise the advantages of the proposed memetic approach over the version [25], we present in Fig. 3 the performance trend of OSNPS and AOSNPS. The trends clearly show the superiority of the newly proposed AOSNPS throughout the entire evolution.

![Performance Trend](image)

**Fig. 3.** Average performance trend (best cost detected $G_{bf}$ over the generations) for OSNPS and AOSNPS.

Finally, in order to illustrate how the concept of convergence can be exported to the context of P systems, we plotted the average Hamming distance $D_{hm}$ calculated over all binary strings produced by the $H$ ESNPSs at each generation $gen$:

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

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; $H$ is the number of individuals (number of ESNPSs); the symbol $\oplus$ represents the OR operator.

It can be noticed that a large value of $D_{hm}$ indicates a high variety between each pair of (binary) candidate solutions in a population. The plot in Fig. 4 shows that AOSNPS achieves and maintains much higher population diversity values than OSNPS. Higher diversity values appear to be beneficial to overcome suboptimal basins of attraction and enhance upon the current best solution.

![Average Hamming Distance](image)

**Fig. 4.** Average Hamming distance $D_{hm}$ (measurement of the population diversity) for OSNPS and AOSNPS.

**V. CONCLUSION**

This paper proposes a novel MA generated by the combination multiple spiking neural P systems and an adaptive Evolutionary Guider Algorithm for solving the 0/1 knapsack problem. Each neural P systems generates a (binary) candidate solution on the basis of the probabilities encoded in its neurons while the Evolutionary Guider Algorithm supervises the search by optimising the functioning of the spiking neural P systems.

The resulting algorithm, Adaptive Optimisation Spiking Neural P Systems (AOSNPS), has been thoroughly tested over multiple problem instances and against various algorithms used to solve the 0/1 knapsack problems. Numerical results show that this unconventional approach is indeed powerful and...
is able to outperform its competitors for most of the problems under consideration. Furthermore, a direct comparison with another algorithm based on a similar structure shows the superiority of AOSNPS in terms of both solution detected and population diversity.

A further important contribution of this study is the novel memetic logic that successfully combines theoretical computer science and metaheuristic optimisation.

ACKNOWLEDGMENT

This work was supported by the National Natural Science Foundation of China (61972324, 61672437, 61702428), by the Sichuan Science and Technology Program (2017FWZ0100, 2018GZ0185, 2018GZ0086), New Generation Artificial Intelligence Science and Technology Major Project of Sichuan Province (2018GZDZX0044) and Artificial Intelligence Key Laboratory of Sichuan Province (2019RYJ06).

REFERENCES


