Journal of Zhejiang University-SCIENCE C (Computers & Electronics)
ISSN 1869-1951 (Print); ISSN 1869-196X (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

A membrane-inspired algorithm with a memory mechanism for knapsack problems*

Juan-juan HE†1, Jian-hua XIAO[2], Xiao-long SHI[1], Tao SONG†‡1

([1]*Key Laboratory of Image Processing and Intelligent Control, School of Automation, Huazhong University of Science and Technology, Wuhan 430074, China*)

([2]*The Logistics Research Center, Nankai University, Tianjin 300071, China*)

†E-mail: hejuanjuan1117@gmail.com; songtao0608@hotmail.com

Received Jan. 3, 2013; Revision accepted June 17, 2013; Crosschecked July 12, 2013

Abstract: Membrane algorithms are a class of distributed and parallel algorithms inspired by the structure and behavior of living cells. Many attractive features of living cells have already been abstracted as operators to improve the performance of algorithms. In this work, inspired by the function of biological neuron cells storing information, we consider a memory mechanism by introducing memory modules into a membrane algorithm. The framework of the algorithm consists of two kinds of modules (computation modules and memory modules), both of which are arranged in a ring neighborhood topology. They can store and process information, and exchange information with each other. We test our method on a knapsack problem to demonstrate its feasibility and effectiveness. During the process of approaching the optimum solution, feasible solutions are evolved by rewriting rules in each module, and the information transfers according to directions defined by communication rules. Simulation results showed that the performance of membrane algorithms with memory cells is superior to that of algorithms without memory cells for solving a knapsack problem. Furthermore, the memory mechanism can prevent premature convergence and increase the possibility of finding a global solution.

Key words: Membrane algorithm, Memory mechanism, Knapsack problem
doi:10.1631/jzus.C1300005 **Document code:** A **CLC number:** TP301

1 Introduction

Membrane computing, a new branch of natural computing, has become a hot topic in recent years. The computing models investigated in membrane computing (usually called P systems) are now known as a group of bio-inspired distributed and parallel computing models. They are abstracted from the structure and functions of living cells, as well as the organization of cells in tissues and organs, including the human brain (Pan and Păun, 2009; Niu *et al.*, 2011; Pan *et al.*, 2011a; Zhang *et al.*, 2011). Until now, P systems have produced many theoretical results and had many applications, such as computing functions (Zhang *et al.*, 2010), computing/generating sets of natural numbers (Zhang *et al.*, 2009a), generating languages (Zhang *et al.*, 2009b), as well as solving practical problems (Zhao and Wang, 2011; Yang and Wang, 2012a).

P systems with cell reproduction can (theoretically) generate exponential work space, thus making it possible to trade space for time, and provide a way to solve computationally hard problems in feasible (polynomial or linear) time. Ishdorj *et al.* (2010) solved two well-known PSPACE-complete problems by spiking neural P systems using pre-computed resources. Neuron division and budding are introduced

‡ Corresponding author

* Project supported by the National Natural Science Foundation of China (Nos. 61033003, 91130034, 61100145, 60903105, and 61272071), the PhD Programs Foundation of the Ministry of Education of China (Nos. 20100142110072 and 2012014213008), and the Natural Science Foundation of Hubei Province, China (No. 2011CDA027)

into spiking neural P systems such that exponential work space can be generated in an efficient way. Spiking neural P systems with neuron division and budding can solve computationally hard problems, such as the satisfiability problem (Pan *et al.*, 2011b). Spiking neural P systems with weights were considered by Wang *et al.* (2010). Spiking neural P systems consisting of two kinds of neurons, and time-free spiking neural P systems have proved to be Turing universal by Pan and Păun (2010) and Pan *et al.* (2011c), respectively. The above solutions were given in an exact way at the theoretical level. However, it is useful to have approximate optimum solutions for realistic problems. Membrane computing has inspired some operations that have been proved to be useful for algorithms finding approximate optimum solutions. Algorithms with operations inspired by membrane computing are called membrane algorithms.

The membrane algorithm was initiated for the traveling salesman problem (Nishida, 2006). The results showed that such algorithms are rather efficient in solving that problem. In recent years, many kinds of membrane algorithms have been presented for various problems and have proved to be successful in finding optimal solutions. Inspired by the hierarchy of a living cell, three membrane algorithms were constructed to deal with single-objective problems (Huang and Wang, 2006), multi-objective numerical optimization problems (Huang *et al.*, 2007), and dynamic multi-objective optimization problems (Huang *et al.*, 2011), respectively. Combining differential evolution algorithms with tissue P systems, another membrane algorithm was proposed for solving manufacturing parameter optimization problems (Zhang *et al.*, 2013). Many membrane algorithms have been proposed for solving practical problems, such as broadcasting problems (Zhang *et al.*, 2012a), image processing (Zhang *et al.*, 2012b), and parameter estimation for fluid catalytic cracking unit (FCCU) reactor-regenerator models (Yang and Wang, 2012b). The results reported show their excellent performance.

Zhang *et al.* (2008) used knapsack problems to compare two frameworks of membrane algorithms, the nested framework and the one-level framework. The simulation results showed that both frameworks are efficient in solving computationally hard problems, but that a membrane algorithm with a one-level framework performs better than a membrane algorithm using a nested framework. Membrane algorithms with a one-level framework can increase the population diversity at the beginning of the computation. As the process continues, the best found solution is used as a start point for further searching. In this way, feasible solutions easily converge on a single point once a better solution is found. This property can provide an easy way to converge to local optima but fails to enhance the capacity of global searching.

In this work, inspired by the fact that neurons in the human brain can work in a distributed manner to process and store information, we consider a variant of a membrane algorithm with a memory mechanism, called 'membrane algorithm with memory mechanism' (RMA). The RMA uses quantum-inspired bit (Q-bit) to represent individuals in a population. Unlike binary, numeric, or symbolic representation, Q-bit, as a probabilistic representation, can represent a linear superposition of states (binary solutions). In this way, a Q-bit individual expresses more than one state, thereby providing more population diversity than other representations. The algorithm consists of two kinds of modules, computation and memory modules, in which cells are arranged in a circular topology and can store and process information, as well as exchange information among themselves. The computation modules are used to explore the search space by using Q-gate as an updating operator. Memory processors are used as local memories to form a stable network retaining the best positions found so far. The process of searching for the optimum solution is determined by various rules in different cells, where feasible solutions denoted by Q-bit are evolved by the rewriting rule in each cell. The information transfers among different cells according to directions defined by the communication rules. We carried out a number of simulations to test the performance of the RMA by solving knapsack problems with different items, and compared our results with those from using a typical genetic algorithm and other two Q-bit representation algorithms. Simulation results showed that the RMA performed well compared to the typical genetic algorithm. It also performed well in terms of stability and its ability to search for solutions compared to a traditional quantum-inspired evolutionary algorithm and another membrane algorithm with a one-level framework, also using Q-bit representation.

2 A membrane algorithm for solving knapsack problems

A knapsack problem is a typical combinatorial optimization problem, and can be formulated as Eqs. (1) and (2). Given a set of m items and a knapsack with a weight limitation, with each item having a weight and a value, maximize the profit function

$$\mathrm{fit}(x)=\sum_{k=1}^{m} p_k x_k \tag{1}$$

subject to

$$\sum_{k=1}^{m} w_k x_k \le C, \tag{2}$$

where p_k, w_k, and C are non-negative integers, p_k is the profit of the kth item, w_k is the weight of item k, and C is the capacity of the knapsack, $x_k \in \{0, 1\}$, for k=1, 2, …, m, and m is the number of items. If x_k=1, the kth item is selected for the knapsack; x_k=0 means that the kth item is not selected for the knapsack. In the following, we introduce an algorithm for solving knapsack problems. We first introduce the membrane structure of the RMA, and then present the procedure and rules.

2.1 Membrane structure

With the aim of preventing premature convergence to increase the possibility of finding a global solution, a more restrictive communication topology can be used to control the speed of information propagation in algorithms. Using more restrictive communication to slow down the convergence speed of algorithms is not new; many restrictive communication topologies have proved to be effective, such as ring, star, or von Neumann (Li, 2010). Considering premature convergence of membrane algorithms with a one-level framework, we use the ring topology to connect two kinds of cells, computation cells (modules) and memory cells (modules), in a particular communication manner. Every computation cell, labeled 1, 2, …, num, has its relative memory cell, labeled m_1, m_2, …, m_{num}, respectively. Each computation cell i interacts only with its immediate memory neighbors m_{i-1} and m_{i+1}, and its relative memory cell m_i, but no computation cells. Each memory cell m_i also connects two neighboring computation cells $i-1$, $i+1$ and its corresponding computation cell i. Three examples of this ring topology are shown in Fig. 1, with num=2, 4, or 6.

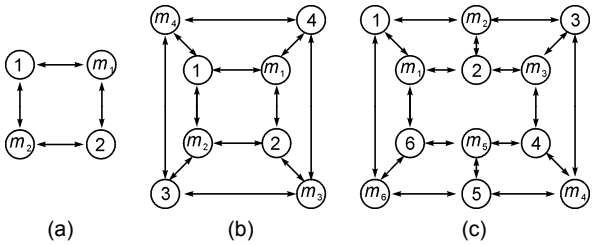

Fig. 1 Three examples of the ring topology

(a) A conventional ring topology using no local memory and with the number of computation cells num=2; (b) The simplest structure using local memory, with num=4; (c) A more complicated structure with num=6

The memory cells can be used as 'anchor' points, providing the best solutions found so far. Meanwhile, each of the feasible solutions can be further improved by the computation cell. The updating processes occur only in computation cells. In each computation cell, a population of solutions is updated and outputs the best current solution. Then, the memory cells choose the best solution among the three solutions obtained in the neighboring computation cells and the solution maintained in them. A computation cell can be updated only if the next generation is the best one within its neighborhood, and has not been improved upon by a better neighbor. So, this is desirable for not only forming a stable network, but also slowing down the speed of convergence to prevent premature convergence.

The best solution in the neighborhood of the ith point is the same as those of its two immediate neighboring members, $(i-1)$th and $(i+1)$th points, but differs from those of members in the neighborhoods further out (Fig. 2). Since the individuals search in their local neighborhood instead of throughout the population, the feasible solutions in different memory cells can differ. Searching for solutions in this way, this ring framework with two types of cells can help maintain better population diversity, and provides a

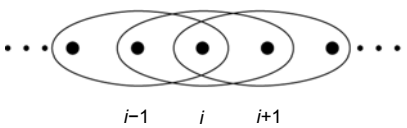

Fig. 2 The communication strategy

Each member communicates with its two immediate neighbors (left and right)

mechanism to slow down information propagation in the population.

2.2 Procedure of the RMA

Before describing the procedure of the RMA, the definitions of Q-bit and Q-gate are addressed briefly in the following. More details can be found in Hey (1999).

Definition 1 (Q-bit) A Q-bit is the smallest unit of quantum information. It is defined by a pair of numbers (α, β) as

$$\begin{bmatrix}\alpha\\ \beta\end{bmatrix}, \tag{3}$$

where $|\alpha|^2+|\beta|^2=1$. $|\alpha|^2$ is the probability that the corresponding bit will be set to '0' state. $|\beta|^2$ is the probability that the corresponding bit will be set to '1' state.

A Q-bit individual can be represented as a string of Q-bit:

$$\boldsymbol{q}=\begin{bmatrix}\alpha_1 & \alpha_2 & \dots & \alpha_m\\ \beta_1 & \beta_2 & \dots & \beta_m\end{bmatrix}, \tag{4}$$

where $|\alpha_k|^2+|\beta_k|^2=1$, k=1, 2, …, m. A Q-bit individual represents a linear superposition of states. Observing a Q-bit individual, a binary sequence is produced probabilistically. Here is an example of an individual with three Q-bits:

$$\begin{bmatrix}1 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}}\\ 0 & \frac{1}{\sqrt{2}} & \frac{\sqrt{6}}{3}\end{bmatrix}. \tag{5}$$

The states of (5) can be represented as: $1/6|000\rangle+1/3|001\rangle+1/6|010\rangle+1/3|011\rangle+0|100\rangle+0|101\rangle+0|110\rangle+0|111\rangle$. This means that an individual with three Q-bits is able to represent the information of eight states. Specifically, the binary sequences {000}, {001}, {010}, {011}, {100}, {101}, {110}, {111} are obtained with probabilities 1/6, 1/3, 1/6, 1/3, 0, 0, 0, 0, respectively. Therefore, Q-bit representation can maintain population diversity.

Definition 2 (Q-gate) A Q-gate is defined as a variation operator of an algorithm with Q-bit representation. The following rotation gate is used in the RMA:

$$\boldsymbol{G}=\begin{bmatrix}\cos(\Delta\theta_k) & -\sin(\Delta\theta_k)\\ \sin(\Delta\theta_k) & \cos(\Delta\theta_k)\end{bmatrix}, \tag{6}$$

where k=1, 2, …, m. A Q-bit can be evolved by a Q-gate as $\boldsymbol{q}'=\boldsymbol{G}\times\boldsymbol{q}$, and the Q-gate rotation angle $\Delta\theta$ decides the corresponding Q-bit toward either the '0' or '1' state.

The procedure flow diagram is shown in Fig. 3.

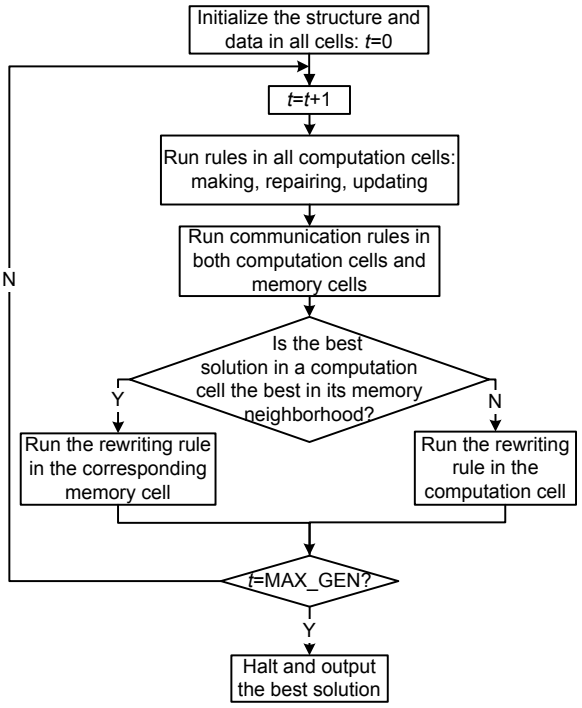

Fig. 3 Main loop of the RMA

The main steps of the procedure are summarized as follows:

Step 1: Initialize the structure of the algorithm and parameters, including num (the number of each kind of module), n (the number of individuals in each computation module), m (the number of items), t (the current iteration number), and MAX_GEN (the maximum number of generations). The individuals in the population of the potential solution are encoded by Q-bit using the following equation:

$$\boldsymbol{q}=\begin{bmatrix}\alpha_1 & \alpha_2 & \dots & \alpha_m\\ \beta_1 & \beta_2 & \dots & \beta_m\end{bmatrix}, \tag{7}$$

where $|\alpha_k|^2+|\beta_k|^2=1$, k=1, 2, …, m. To observe a feasible solution $\boldsymbol{x}=(x_1, x_2, \ldots, x_m)$ from such a Q-bit individual, a random number r is generated from the range (0, 1). If $r<|\beta_k|^2$, the corresponding bit x_k is set to 1; this means that the kth item is selected for the knapsack. Otherwise, x_k is set to 0.

Step 2: $t=t+1$. If $t<$MAX_GEN, repeat steps 3–5.

Step 3: Start the computation in all computation cells. First, the set of solutions

$$X^t=\{x_{1,1}^t, x_{1,2}^t, \ldots, x_{i,j}^t, \ldots, x_{\mathrm{num},n}^t\}$$

is produced by making rules (Eq. (8)) from observing the states of $Q^{t-1}=\{q_{1,1}^{t-1}, q_{1,2}^{t-1}, \ldots, q_{i,j}^{t-1}, \ldots, q_{\mathrm{num},n}^{t-1}\}$, where i=1, 2, …, num, j=1, 2, …, n. The process is similar in quantum computing. Because every computation cell contains n individuals with m bits, the system generates num×n individuals at the beginning in num×n×m steps. Then, repairing rules (Eq. (9)) are used to make sure that every solution satisfies the capacity. If $\sum_{k=1}^{m} w_k x_k > C$, the knapsack is overfilled. One item is randomly selected from the current knapsack. Continue taking out items, until $\sum_{k=1}^{m} w_k x_k < C$. If $\sum_{k=1}^{m} w_k x_k < C$, the algorithm keeps loading items until the knapsack is overfilled. To satisfy the capacity of the knapsack problem, the last item that is put in the knapsack is taken out. The cost is at least num×n×m steps for calculating the weights of the knapsack and two steps for setting x_i (i is a random number). Afterwards, Q-bit individuals in $Q(t)$ are updated by the updating rules (Eq. (10)). A rotation gate is applied to update Q-bit individuals as in quantum computing. Every bit needs two steps for updating by the rotation gate. The updating rule will cost 2×num×n×m steps.

Step 4: When all the individuals in computation cells finish updating, the communication channels are opened. The copies of solutions in both computation and memory cells are sent into the cells to which they are connected by communication rules (Eq. (13)). The formalization of the communication rule can be expressed as (i, u/v, j), where u, v represent the objects (solutions or a set of solutions) in the searching areas i and j, respectively. This means that the system sends object u from region i to region j, and object v is sent from region j to region i at the same time. Because every cell connects with three neighbors, the communication rules will cost 2×3×num steps.

Step 5: If the best solution in computation cell i is the best within its neighborhood (fit(i)>best{fit(m_{i-1}), fit(m_i), fit(m_{i+1})}), the rewriting rule will be applied in memory cell m_i (Eq. (11)). Otherwise, there exists at least one solution in the neighborhood that is better than the result after updating. This means that the updating process has failed in computation cell i in the current iteration. Then, the rewriting rule will be used in computation cell i (Eq. (12)). The computation cell i is ready for the next generation. The rewriting rule will cost num×n×m steps to calculate the fitness of every feasible solution and 3×num steps for comparison.

Step 6: When t=MAX_GEN, the termination criterion is met. The best solution is sent out of the system.

In the above procedure, the rules of making, repairing, updating, rewriting, and communication are used in the parallel membranes. These rules are described mathematically as follows:

1. Making rule, $\boldsymbol{q}^{t-1}\rightarrow\boldsymbol{x}^t$:

$$\begin{cases} \boldsymbol{q}^{t-1}=\begin{bmatrix} \alpha_1^{t-1} & \alpha_2^{t-1} & \ldots & \alpha_m^{t-1} \\ \beta_1^{t-1} & \beta_2^{t-1} & \ldots & \beta_m^{t-1} \end{bmatrix}, \\ \boldsymbol{x}^t=(x_1^t, x_2^t, \ldots, x_m^t),\ x_k^t\in\{0,1\},\ k=1,2,\ldots,m. \\ \text{If random}[0,1]<|\beta_k^{t-1}|^2,\ \text{then } x_k^t=1; \\ \text{else } x_k^t=0. \end{cases} \quad (8)$$

2. Repairing rule, $\boldsymbol{x}\rightarrow\boldsymbol{x}'$:

$$\begin{cases} \boldsymbol{x}=(x_1, x_2, \ldots, x_m),\ \boldsymbol{x}'=(x_1', x_2', \ldots, x_m'), \\ x_k\in\{0,1\},\ x_k'\in\{0,1\},\ k=1,2,\ldots,m. \\ \text{If } \sum_{k=1}^{m} w_k x_k > C,\ \text{then } x_g'=0,\ \text{and} \\ \{x_1, \ldots, x_{g-1}, x_{g+1}, \ldots, x_m\} \\ \Rightarrow \{x_1', \ldots, x_{g-1}', x_{g+1}', \ldots, x_m'\}; \\ \text{if } \sum_{k=1}^{m} w_k x_k < C,\ \text{then } x_g'=1,\ \text{and} \\ \{x_1, \ldots, x_{g-1}, x_{g+1}, \ldots, x_m\} \\ \Rightarrow \{x_1', \ldots, x_{g-1}', x_{g+1}', \ldots, x_m'\}. \\ g \text{ is an integer and } g=\text{random}[1,m]. \end{cases} \quad (9)$$

3. Updating rule, $\boldsymbol{q}^{t-1}\rightarrow\boldsymbol{q}^{t}$:

$$\begin{cases}\boldsymbol{q}^{t-1}=\begin{bmatrix}\alpha_1^{t-1} & \alpha_2^{t-1} & ... & \alpha_m^{t-1}\\ \beta_1^{t-1} & \beta_2^{t-1} & ... & \beta_m^{t-1}\end{bmatrix},\\ \boldsymbol{q}^{t}=\begin{bmatrix}\alpha_1^{t} & \alpha_2^{t} & ... & \alpha_m^{t}\\ \beta_1^{t} & \beta_2^{t} & ... & \beta_m^{t}\end{bmatrix},\\ \begin{bmatrix}\alpha_k^{t}\\ \beta_k^{t}\end{bmatrix}=\begin{bmatrix}\cos(\Delta\theta_k) & -\sin(\Delta\theta_k)\\ \sin(\Delta\theta_k) & \cos(\Delta\theta_k)\end{bmatrix}\begin{bmatrix}\alpha_k^{t-1}\\ \beta_k^{t-1}\end{bmatrix},\\ k=1,2,...,m.\end{cases} \tag{10}$$

4. Rewriting rule

(1) Rewriting rule in memory cell m_i:

$$x_{m_i}\rightarrow\max\{X_i\},$$

where

$$\begin{cases}x_{m_i}\text{ is the solution in the }i\text{th memory cell},\\ \max\{X_i\}=\max\{x_{i,1},x_{i,2},...,x_{i,n}\},\\ x_{i,j}\text{ is the }j\text{th solution in the }i\text{th computation cell},\\ i=1,2,...,\text{num},\ j=1,2,...,n.\end{cases} \tag{11}$$

(2) Rewriting rule in computation cell i:

$$\min\{X_i\}\rightarrow x_{m_i},$$

where

$$\begin{cases}\min\{X_i\}=\min\{x_{i,1},x_{i,2},...,x_{i,n}\},\\ x_{i,j}\text{ is the }j\text{th solution in the }i\text{th computation cell},\\ x_{m_i}\text{ is the solution in the }i\text{th memory cell},\\ i=1,2,...,\text{num},\ j=1,2,...,n.\end{cases} \tag{12}$$

5. Communication rule, $(p,X_p/x_{m_q},q)$:

$$\begin{cases}X_p\text{ is the set of solutions in the }p\text{th computation cell},\\ x_{m_q}\text{ is the solution in the }q\text{th memory cell},\\ p,q\text{ are integers, }p,q\in[1,\text{num}],\ |p-q|\leq 1.\end{cases} \tag{13}$$

The above rules clearly show how this membrane-inspired algorithm works. The computational complexity of RMA can be represented as O(num×n×m). In the following sections, we will describe extensive simulations to support the above statements. We use RMA to solve a knapsack problem with different items, and compare the results obtained by RMA with those obtained by three other algorithms: a typical genetic algorithm, the traditional quantum-inspired evolutionary algorithm (QEA) (Han and Kim, 2002), and another membrane algorithm with a one-level framework that also uses Q-bit representation (QEPS) (Zhang *et al.*, 2008). To illustrate the efficiency of the framework of the membrane algorithm, the population size of QEA was set to n×m. Therefore, the computational complexity of QEA and QEPS can also be represented as O(num×n×m). Moreover, since both QEA and QEPS use randomly generated instances of knapsack problems to illustrate their performance, we also used such instances to evaluate the algorithm proposed in this study.

3 Simulation results

In all simulations, the following sets of data were considered:

$$w_k\text{, uniformly random in the interval [1, 10]},$$
$$p_k=w_k+5,$$

and the following average knapsack capacity was used:

$$C=\frac{1}{2}\sum_{k=1}^{m}w_k.$$

Three instances of small scale knapsack problems with 150, 300, and 550 items, and five of large scale knapsack problems with 800, 1100, 1400, 1700, and 2000 items were considered. For each scale of problem, a particular instance was used.

3.1 Angle selection

The angle parameters used for the rotation gate were selected from Table 1, and were the same as those described by Han and Kim (2002). The Q-gate rotation angles θ_1, θ_2, …, θ_8 could be selected easily by intuitive reasoning. $\boldsymbol{x}$=(x_1, x_2, …, x_m) is one individual in a computation module. $\boldsymbol{b}$=(b_1, b_2, …, b_m) is the best individual in this population. For example, when x_i=0, b_i=0, and if the condition fit($\boldsymbol{x}$)≥fit($\boldsymbol{b}$) is false, the value of θ_3 is set to a positive value to increase the probability of state 1. If x_i=1, b_i=0, and the

condition fit($\boldsymbol{x}$)≥fit($\boldsymbol{b}$) is false, θ_5 is set to a negative value to increase the probability of state 0. The angle $\Delta\theta_i$ affects the speed of convergence. Fig. 4 shows the results of using the RMA on the knapsack problems with 150, 300, and 550 items to find good parameter settings of θ_3 and θ_5. The number of computation cells was set to 10, and the size of the population was set to 4.

Table 1 Selection of rotation angles θ_1, θ_2, …, θ_8

x_i	b_i	fit($\boldsymbol{x}$)≥fit($\boldsymbol{b}$)	$\Delta\theta_i$
0	0	False	θ_1
0	0	True	θ_2
0	1	False	θ_3
0	1	True	θ_4
1	0	False	θ_5
1	0	True	θ_6
1	1	False	θ_7
1	1	True	θ_8

fit(·) is the function of profit. x_i is the ith bit of the current solution; b_i is the ith bit of the best solution. Here, θ_1=0, θ_2=0, θ_3=$\Delta\theta$, θ_4=0, θ_5=−$\Delta\theta$, θ_6=0, θ_7=0, θ_8=0

Three instances with different items were generated randomly. When simulations were carried out with the same number of items, the identical instance was used. The values of 0.0025π, 0.005π, 0.01π, 0.02π, and 0.05π were tested for θ_3 and $-\theta_5$.

The best profit was obtained over 30 runs. The arithmetic average value was calculated from the best values obtained after running the procedure 30 times. From the results in Fig. 4a, some values for θ_3 and θ_5 can be selected, such as 0.0025π and −0.0025π, 0.005π and −0.005π, and 0.05π and −0.05π. In Figs. 4b and 4c, we can see that the algorithm performed well when θ_3=0.005π and θ_5=−0.005π. The values of 0.005π and −0.005π were chosen for θ_3 and θ_5, respectively, for the following simulations because of the better profits they obtained.

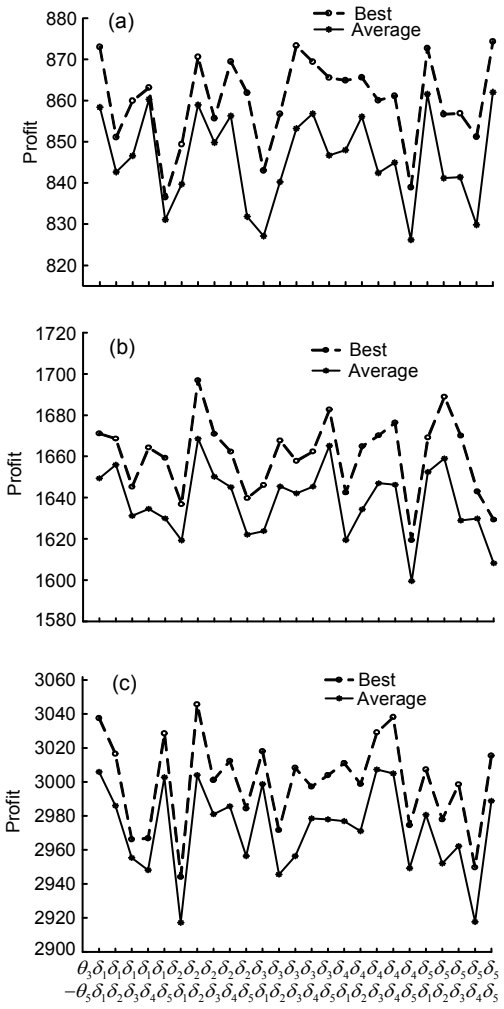

Fig. 4 Best and average profits from using the RMA on knapsack problems with 150 (a), 300 (b), or 550 (c) items to find good parameter settings of θ_3 and θ_5

The number of computation cells was set to 10, and the size of the population was set to 4. δ_1, δ_2, δ_3, δ_4, and δ_5 were 0.0025π, 0.005π, 0.01π, 0.02π, and 0.05π, respectively

3.2 Size of the ring structure

The size of the ring structure is an important parameter of the RMA. As described in Section 2.1, each computation cell has its corresponding memory cell. The number of computation cells (num) can be used to measure the size of the ring structure. The size num affects the balance of diversity and running time. For the diversity, because memory cells record local memories, different solutions can be obtained in every memory cell. As more memory cells are used in the computation, more different solutions are obtained for the next iteration. On the other hand, because the population size of the system is increased with the size of the ring structure, the running time is also increased. To investigate the effect of num on the performance of the RMA, three knapsack problems with 150, 300, and 550 items, respectively, were tested. The number of individuals in each computation cell n was set to 4. For all simulations, when the number of iterations was over 100, the termination criterion was met, because the results increased slightly after the 60th iteration (Fig. 5).

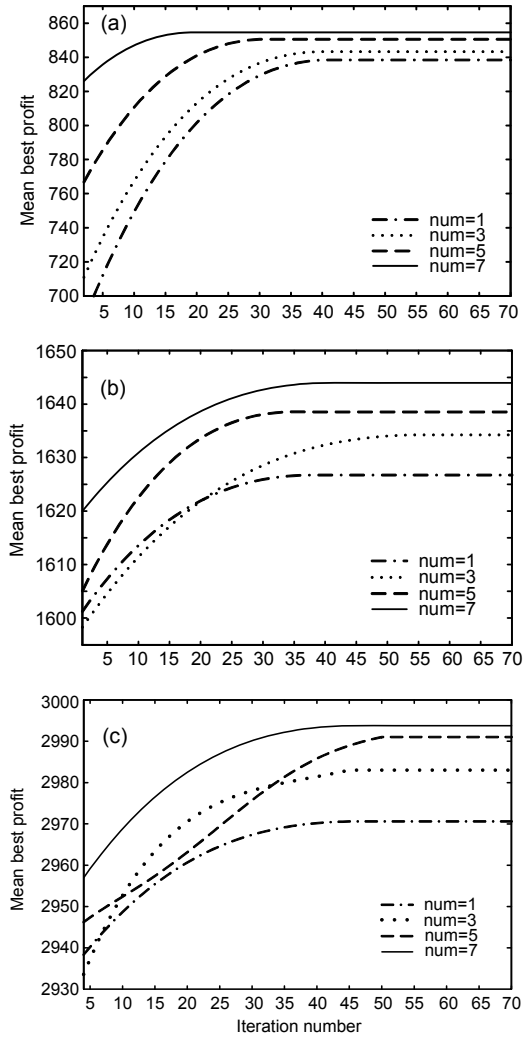

Fig. 5 The progress of the mean best profits of various numbers of computation cells (1, 3, 5, or 7) tested with 150 (a), 300 (b), or 550 (c) items

All the results were obtained over 30 runs with num varying from 1 to 10. Note that, if num=1, the population size of the RMA works as QEA and the information used for updating is obtained only from the best solutions found. When num<4, the RMA works without local memory and is used as QEPS. In addition, the curves in Fig. 5 were obtained by squares polynomial smoothing. The progress of the mean best profit with various values of num indicates that the RMA has a better balance between exploration and exploitation as num increases from 1 to 10. This means that algorithms with more cells maintain better diversity and converge to a better solution quickly.

The results of the simulations on different values of num from 1 to 10 are shown in Fig. 6. The results show that the solutions improve as the number of cells increases, and that the average profits stay at a stable level. The best profits increased with num because the greater the number of cells, the more directions towards the optimal solution can be explored by the RMA. Since the ring topology forms a stable network, the mean best profits also increase with the size of the ring structure. With num increasing, more local optima should be found. That is the reason why the mean profits of all cells stay at a steady level. Considering the running time, we set num=7.

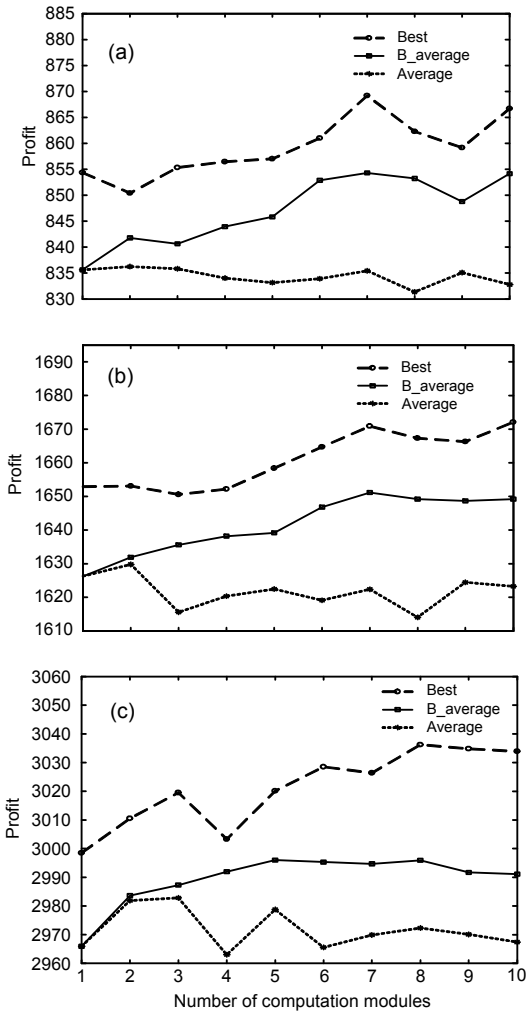

Fig. 6 Simulation results of three knapsack problems with 150 (a), 300 (b), or 550 (c) items

Best, B_average, and Average denote the best solution, mean best solution, and mean solution of all memory cells, respectively

3.3 Number of individuals in a computation cell

Simulations using the RMA for the knapsack problems with 150, 300, and 550 items were carried out to investigate the effect of the number of individuals in a computation cell. The number of computation modules, num, was set to 7, and the number of memory modules was also 7. The number of individuals in each computation module, n, varied from 1 to 10. Results from the application of different values of n are shown in Fig. 7.

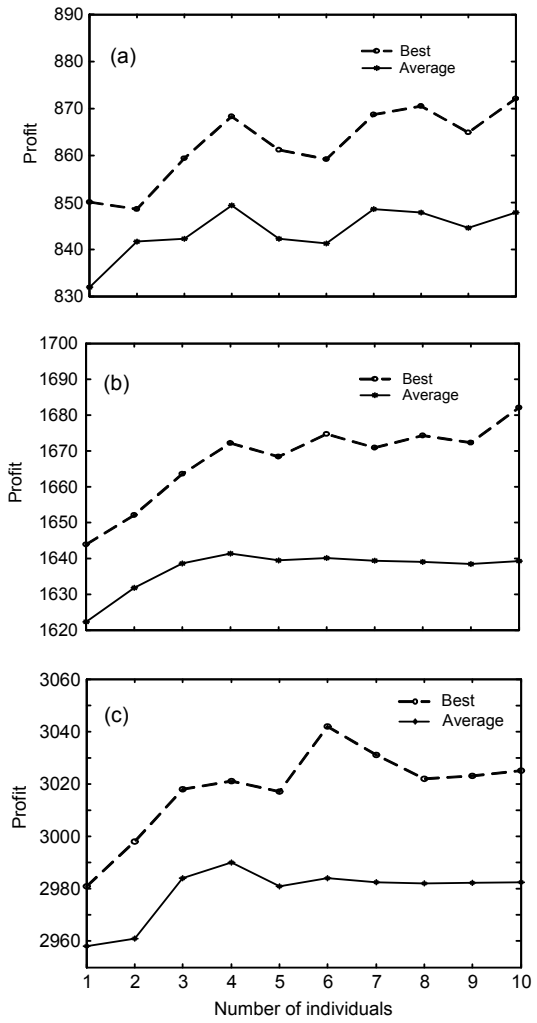

Fig. 7 Best and average profits from using the RMA on the knapsack problem with 150 (a), 300 (b), or 550 (c) items
The results were obtained with 100 iterations over 30 runs

In each simulation, the best profits and the mean best profits over 30 runs increased as the number of cells increased from 1 to 4, while the number of solutions increased little with more individuals when n>4 (Fig. 7). When n>4, the values of fitness were almost at the same level, except that the best profit was obtained when n=6 (Fig. 7c). Considering the length of running time, n=4 was chosen in this study.

4 Comparison with other algorithms

To evaluate the effectiveness of this ring topology with two kinds of cells, we compared our algorithm with three other algorithms—a typical genetic algorithm, the traditional QEA and QEPS. QEPS is another quantum-inspired evolutionary algorithm based on membrane computing, and has a one-level structure. Eight instances with different items were generated randomly. When simulations were carried out with the same number of items, the identical instance was used. Simulation results over 30 runs with eight different items were collected, and summarized in Table 2 (num=7, n=4).

The RMA yielded better results than GA, QEA, or QEPS. The results were also more stable, with smaller standard deviations, even when solving the larger knapsack problem. This may have been because QEA is a type of stochastic optimization algorithm requiring several operations, and the results of its solutions have poor stability. Moreover, QEA easily converges to a local optimum because of premature convergence. Compared with traditional QEA with a single processor, the RMA increases the number of individuals, which will increase the running time, as with QEPS.

5 Conclusions

In this work, a membrane-inspired algorithm is proposed. Inspired by the brain's information processing, we use two types of cells, computation cells and memory cells, in a particular communication framework. In this framework, the RMA is designed for solving knapsack problems with different numbers of items. According to the simulation results, we have demonstrated that the RMA is able to induce stable behavior and perform well in solving knapsack problems.

Table 2 Simulation results for the knapsack problem*

Number of items	Method	BS	MBS	WS	STD
150	GA	845.8	838.5	820.6	7.2
	QEA	844.0	831.9	820.5	6.9
	QEPS	846.8	834.2	825.2	6.4
	RMA	854.2	847.4	838.9	4.5
300	GA	1652.6	1625.0	1606.0	10.5
	QEA	1659.8	1639.6	1616.4	10.0
	QEPS	1654.5	1638.8	1624.4	10.2
	RMA	1672.8	1661.1	1654.3	5.0
550	GA	2998.2	2972.2	2942.7	16.8
	QEA	2993.5	2965.0	2941.8	14.9
	QEPS	2995.3	2967.9	2941.2	12.3
	RMA	3012.4	2990.7	2978.3	8.5
800	GA	4291.1	4263.2	4233.0	18.5
	QEA	4304.5	4263.5	4240.7	17.9
	QEPS	4307.8	4269.1	4244.6	16.8
	RMA	4325.2	4296.1	4278.9	10.3
1100	GA	5897.8	5856.0	5819.4	19.9
	QEA	5891.3	5864.3	5832.6	19.6
	QEPS	5910.3	5861.6	5836.5	18.7
	RMA	5925.3	5902.8	5885.9	9.9
1400	GA	7529.5	7491.1	7443.1	30.1
	QEA	7544.6	7496.4	7465.3	19.7
	QEPS	7556.2	7493.3	7452.6	20.4
	RMA	7579.5	7541.1	7518.3	13.2
1700	GA	9161.2	9115.5	9069.2	23.5
	QEA	9134.8	9107.6	9057.0	23.3
	QEPS	9165.2	9109.5	9074.7	21.6
	RMA	9211.9	9170.2	9124.8	17.3
2000	GA	10715	10649	10574	30.7
	QEA	10714	10652	10591	28.6
	QEPS	10727	10651	10601	27.9
	RMA	10753	10681	10648	19.1

* The maximum number of generations was 100, the number of runs was 30, num=7, and n=4. BS, MBS, WS, and STD represent best solution, mean best solution, worst solution, and standard deviation, respectively

In the RMA, Q-bit representation and a Q-gate updating process are used. Previous algorithms using Q-gate as an updating operator (including QEA and QEPS) easily converge to a local optimum. To avoid premature convergence in the evolution, we introduce a memory mechanism in the algorithm, inspired by the function of biological neuron cells storing information. The memory mechanism can maintain several local optima and slows down the speed of convergence. Moreover, instead of using the information from the current best individual for the updating process, RMA uses several good individuals and keeps multiple directions for expanding the searching area. In this way, the influence of the initial population can be reduced and stable behavior is induced. The simulation results show that the RMA has greater stability than other algorithms.

Understanding the computations going on in nature, using this knowledge to obtain more efficient algorithms or even new types of computers is a challenging and promising research topic. In biology, over 300 different kinds of cells have been found in living organisms for performing specific tasks. A wider range of cells or communication mechanisms should be tried to discover new ideas, tools, techniques, and models for improving algorithms.

Many real-world problems are 'multimodal' by nature. That is, they have multiple global and local optima. It might be desirable to locate not only the global optima but also some local optima that are considered as being satisfactory. In our method, solutions obtained in memory cells are different from each other. Therefore, the algorithm can work on problems with multiple global and local optima. Moreover, many realistic optimization problems, such as bridge construction, aircraft design, and chemical plant design, require the simultaneous optimization of more than one objective function. We hope that the proposed algorithm can be used to solve multi-objective optimization. In addition, the RMA was implemented on a traditional computer in this study. The advantages of the ring topology with its particular communication mechanism would be more obvious when implemented on a distributed system. If the procedure were applied in distributed parallel processing hardware, the computational complexity would be $O(n\times m)$.

References

Han, K.H., Kim, J.H., 2002. Quantum-inspired evolutionary algorithm for a class of combinatorial optimization. *IEEE Trans. Evol. Comput.*, **6**(6):580-593. [doi:10.1109/TEVC.2002.804320]

Hey, T., 1999. Quantum computing: an introduction. *Comput. Control Eng. J.*, **10**(3):105-142. [doi:10.1049/cce:19990303]

Huang, L., Wang, N., 2006. An optimization algorithm inspired by membrane computing. *LNCS*, **4222**:49-52. [doi:10.1007/11881223_7]

Huang, L., He, X., Wang, N., Xie, Y., 2007. P systems based multi-objective optimization algorithm. *Progr. Nat. Sci.*, **17**(4):458-465. [doi:10.1080/10020070708541023]

Huang, L., Suh, I.H., Abranham, A., 2011. Dynamic multi-objective optimization based on membrane computing for control of time-varying unstable plants. *Inf. Sci.*, **181**(11): 2370-2391. [doi:10.1016/j.ins.2010.12.015]

Ishdorj, T.O., Leporati, A., Pan, L., Zeng, X., Zhang, X., 2010. Deterministic solutions to QSAT and Q3SAT by spiking neural P systems with pre-computed resources. *Theor. Comput. Sci.*, **411**(25):2345-2358. [doi:10.1016/j.tcs.2010.01.019]

Li, X., 2010. Niching without niching parameters: particle swarm optimization using a ring topology. *IEEE Trans. Evol. Comput.*, **14**(1):150-169. [doi:10.1109/TEVC.2009.2026270]

Nishida, T.Y., 2006. Membrane algorithms. *LNCS*, **3850**:55-66. [doi:10.1007/11603047_4]

Niu, Y., Pan, L., Pérez-Jiménez, M.J., Font, M.R., 2011. A tissue systems based uniform solution to tripartite matching problem. *Fundam. Inf.*, **109**(2):179-188.

Pan, L., Păun, G., 2009. Spiking neural P systems with anti-spikes. *Int. J. Comput. Commun. Control*, **4**(3):273-282.

Pan, L., Păun, G., 2010. Spiking neural P systems: an improved normal form. *Theor. Comput. Sci.*, **411**(6):906-918. [doi:10.1016/j.tcs.2009.11.010]

Pan, L., Daniel, D.P., Pérez-Jiménez, M.J., 2011a. Computation of Ramsey numbers by P system with active membranes. *Int. J. Found. Comput. Sci.*, **22**(1):29-38. [doi:10.1142/S0129054111007800]

Pan, L., Păun, G., Pérez-Jiménez, M.J., 2011b. Spiking neural P systems with neuron division and budding. *Science China Inf. Sci.*, **54**(8):1596-1607. [doi:10.1007/s11432-011-4303-y]

Pan, L., Zeng, X., Zhang, X., 2011c. Time-free spiking neural P systems. *Neur. Comput.*, **23**(5):1320-1342. [doi:10.1162/NECO_a_00115]

Wang, J., Hoogeboom, H.J., Pan, L., Păun, G., Pérez-Jiménez, M.J., 2010. Spiking neural systems with weights. *Neur. Comput.*, **22**(10):2615-2646. [doi:10.1162/NECO_a_00022]

Yang, S., Wang, N., 2012a. A novel P systems based optimization algorithm for parameter estimation of proton exchange membrane fuel cell model. *Int. J. Hydr. Energy*, **37**(10):8465-8476. [doi:10.1016/j.ijhydene.2012.02.131]

Yang, S., Wang, N., 2012b. A P systems based optimization algorithm for parameter estimation of FCCU reactor-regenerator model. *Chem. Eng. J.*, **211-212**:508-518. [doi:10. 1016/j.cej.2012.08.040]

Zhang, G., Gheorghe, M., Wu, C., 2008. A quantum-inspired evolutionary algorithm based on P systems for knapsack problem. *Fundam. Inf.*, **87**(1):93-116.

Zhang, G., Zhou, F., Huang, X., Cheng, J., Gheorghe, M., Ipate, F., Lefticaru, R., 2012a. A novel membrane algorithm based on particle swarm optimization for solving broadcasting problems. *J. Univ. Comput. Sci.*, **18**(13):1821-1841. [doi:10.3217/jucs-018-13-1821]

Zhang, G., Gheorghe, M., Li, Y., 2012b. A membrane algorithm with quantum-inspired subalgorithms and its application to image processing. *Nat. Comput.*, **11**(4):701-717. [doi:10.1007/s11047-012-9320-2]

Zhang, G., Cheng, J., Gheorghe, M., Meng, Q., 2013. A hybrid approach based on differential evolution and tissue membrane systems for solving constrained manufacturing parameter optimization problems. *Appl. Soft Comput.*, **13**(3):1528-1542. [doi:10.1016/j.asoc.2012.05.032]

Zhang, X., Wang, J., Pan, L., 2009a. A note on the generative power of axon systems. *Int. J. Comput. Commun. Control*, **4**(1):92-98.

Zhang, X., Zeng, X., Pan, L., 2009b. On languages generated by asynchronous spiking neural P systems. *Theor. Comput. Sci.*, **410**(26):2478-2488. [doi:10.1016/j.tcs.2008.12.055]

Zhang, X., Jiang, Y., Pan, L., 2010. Small universal spiking neural P systems with exhaustive use of rules. *J. Comput. Theor. Nanosci.*, **7**(5):890-899. [doi:10.1166/jctn.2010.1436]

Zhang, X., Wang, S., Niu, Y., Pan, L., 2011. Tissue P systems with cell separation: attacking the partition problem. *Sci. China Inf. Sci.*, **54**(2):293-304. [doi:10.1007/s11432-010-4162-y]

Zhao, J., Wang, N., 2011. A bio-inspired algorithm based on membrane computing and its application to gasoline blending scheduling. *Comput. Chem. Eng.*, **35**(2):272-283. [doi:10.1016/j.compchemeng.2010.01.008]