A Modified Membrane-Inspired Algorithm Based on Particle Swarm Optimization for Mobile Robot Path Planning

X.Y. Wang, G.X. Zhang, J.B. Zhao, H.N. Rong, F. Ipate, R. Lefticaru

Xueyuan Wang
1. School of Electrical Engineering
Southwest Jiaotong University
Chengdu 610031, P.R. China
2. School of Information Engineering
Southwest University of Science and Technology
MianYang 621010, P.R.China
121053406@qq.com

Gexiang Zhang*, Junbo Zhao, Haina Rong
School of Electrical Engineering
Southwest Jiaotong University
Chengdu 610031, P.R. China
gexiangzhang@gmail.com, junbozhao55589@gmail.com, ronghaina@126.com
*Corresponding author

Florentin Ipate, Raluca Lefticaru
Faculty of Mathematics and Computer Science
University of Bucharest
Academiei 14, Bucharest, Romania
florentin.ipate@ifsoft.ro, raluca.lefticaru@fmi.unibuc.ro

Abstract

To solve the multi-objective mobile robot path planning in a dangerous environment with dynamic obstacles, this paper proposes a modified membrane-inspired algorithm based on particle swarm optimization (mMPSO), which combines membrane systems with particle swarm optimization. In mMPSO, a dynamic double one-level membrane structure is introduced to arrange the particles with various dimensions and perform the communications between particles in different membranes; a point repair algorithm is presented to change an infeasible path into a feasible path; a smoothness algorithm is proposed to remove the redundant information of a feasible path; inspired by the idea of tightening the fishing line, a moving direction adjustment for each node of a path is introduced to enhance the algorithm performance. Extensive experiments conducted in different environments with three kinds of grid models and five kinds of obstacles show the effectiveness and practicality of mMPSO.

Keywords: Membrane computing, evolutionary membrane computing, particle swarm optimization, variable dimensions, mobile robot path planning, membrane systems.

1 Introduction

As a branch of natural computing, membrane computing(MC), initiated by Păun in 1998, aims to abstract distributed and parallel computing models, also called P systems or membrane systems, from the compartmentalized structure and interactions of living cells [1, 2]. There are three main research directions in this area [3]: theoretical study including computing models

and their computing power and efficiency; applications such as modeling biological processes and approximately solving engineering optimization problems [4]; software and hardware realization. In the past fifteen years, much attention has been paid to the theoretical aspects, but the applications are worth further discussing, especially for solving real-world engineering problems.

Evolutionary algorithms (EAs) are a class of probabilistic search methods with many advantages such as flexibility, convenient application and robustness. While MC can provide flexible evolution rules and parallel-distributed framework [5], which is very beneficial to produce the membrane-inspired evolutionary algorithms (MIEAs). Until now, different kinds of MIEAs have been proposed. In [6], a certain number of nested membrane structures in the skin membrane were combined with EAs for multi-objective optimization problems. A novel MIEA, called QEPS, combining quantum-inspired evolutionary algorithms with P systems to solve image processing problems and knapsack problems were proposed in [7, 8]. Particle swarm optimization (PSO) [9] with one-level membrane structure (OLMS) was used to solve broadcasting problems of P systems and radio spectrum allocation, respectively [10, 13]. In [11], Xiao et al. applied the bio-inspired algorithm based on membrane computing for engineering design problems.

These investigations prove the usefulness of introducing the P systems into EAs in order to solve various real-world applications. To the best of our knowledge, there is not any work focusing on the use of a MIEA to solve mobile robot path planning problems (MR3P), which is a very important real-world application.

In this paper, a modified membrane-inspired algorithm based on particle swarm optimization (mMPSO) is proposed to solve MR3P. The main contributions of this paper can be summarized as follows: (1) In this study, the solution to MR3P is considered as a dimension-reducing optimization problem and therefore a PSO with variable dimensions (vPSO) is introduced into mMPSO, and further a dynamic double OLMS (D-OLMS) with membrane division and dissolution is presented; this is combined with vPSO to arrange the particles and execute the communications between regions delimited by membranes. (2) Mobile robot path planning is a multi-objective optimization problem. This study considers three objectives, *distance*, *safety* and *smoothness*, instead of a single objective (path length) [15–17] or bi-objectives (path length and risk degree) as previously considered in the literature [18, 21]. A point repair algorithm and a smoothness approach are presented to effectively trade-off multiple objectives and speed-up the mMPSO convergence. (3) Inspired by the idea of tightening the fishing line, a moving direction adjustment for each node of a path is introduced to enhance the algorithm performance, together with the point repair algorithm. (4) Extensive experiments are carried out by considering various environments with different grid models and different obstacles to verify the effectiveness and practicality of mMPSO.

The rest of this paper is organized as follows. Section 2 describes MR3P. In Section 3, we present mMPSO for solving MR3P. Section 4 discusses parameter setting and provides experimental results. Conclusions are drawn in Section 5.

2 Mobile Robot Path Planning Problems

This section gives a brief description of MR3P and then summarizes the related works.

2.1 Problem Statement

MR3P aims to find a reasonable collision-free path for a mobile robot from the starting position to the target position through an environment containing static or dynamic obstacles. It is proved that MR3P is an NP-complete problem [19]. Mobile robots are very useful for dangerous or hostile environments that humans are not able to reach. As one of the important

research themes in the mobile robotics field, MR3P, launched in the 1960s, has become an attractive and important research area.

Generally, the criterion for planning a mobile path has to consider many factors, such as the shortest distance, safety degree, smoothness, the lowest energy cost and minimum time, based on the characteristics of a special robot with the minimal turning radius, acceleration and the limited velocity and the features of the environment, such as the distances between obstacles, the shapes of obstacles, the occurrence probabilities of dynamic obstacles. Thus, the optimization of a mobile robot path in a static or dynamic environment is very complicated. For implementing the real time robot path planning in a dynamic environment or for dynamically tracking the motion target of a robot, at least three aspects should be carefully considered. These are also strongly interconnected.

(I) An efficient and effective optimization approach is very important for planning a good mobile robot path. Aiming to solve MR3P in a dynamic environment, this study proposes mMPSO.

(II) A simple and good objective function is very important for planning a good mobile robot path. In this study, the objective function aims to minimize the path length, to maximize the smoothness and the distances between a robot and the obstacles or dangerous sources, and can be expressed as

$$f = K_d \cdot Dis + K_f \cdot S + K_s \cdot SD \tag{1}$$

where K_d, K_s, K_v are the weighing factors of the path length, smoothness and safety degree, respectively. The detailed description of path length, smoothness and safety degree are as follows:

1. ***Path length***: Path length Dis is the sum of distances between n nodes from the starting point to the end point and can be described as

$$Dis = \sum_{i=0}^{n-1} L\left(i, i+1\right) \tag{2}$$

where $L(i, i+1) = \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}$ is the distance between nodes i and $i+1$, where x_i and x_{i+1} are the x-axis values of nodes i and $i+1$; y_i and y_{i+1} are the y-axis values of nodes i and $i+1$, respectively.

2. ***Smoothness***: Smoothness refers to the sum of the reflection angles formed by any three neighboring nodes of a path. As usually calculating directly the smoothness is a time-consuming process, this study uses an indirect approach, i.e., it uses the ratio S_c of the number of deflection angles which are less than the given expected value to the total number of deflection angles and the ratio S_p of the number of path segments more than the number of the segments in the path with the smallest number of path segments in a group to the total number of path segments to evaluate the smoothness of a path. Smoothness can be calculated by using the following formula:

$$S = \alpha \cdot S_c + \beta \cdot S_p \tag{3}$$

where $S_c = 1 - \frac{DA_l}{N_f - 1}$; $S_p = 1 - \frac{S_{min}}{N_f}$, where N_f is the total number of path segments; DA_l is the number of deflection angles greater than the expected value; S_{min} is the number of the segments in the path with the smallest number of path segments in a group; α and β are two weighting coefficients.

3. ***Safety degree***: Safety degree (SD) is the sum of deviation degrees C_i $(i = 1, 2, \ldots, N)$ between any segment in a path and its nearest obstacle. SD is defined as

$$SD = \sum_{i=1}^{n-1} C_i = \begin{cases} 0, & d_i \geq \lambda \\ \sum_{i=1}^{n-1} e^{\lambda - d_i}, & d_i < \lambda \end{cases} \tag{4}$$

where d_i is the minimal distance between the ith segment and its nearest obstacle; λ is the threshold of safty degree.

(III) The establishment of an environment model is the foundation of MR3P and decides the environment feature (static or dynamic), and how to choose an evaluation method and an optimization approach to implement the path planning for a mobile robot. There are three main environment models: vector (obstacles represented by polygons), grid (occupancy cell) and graph (Voronoi diagram or visibility graph). As compared with vector and graph, grid has the advantages of simple and flexible. This study uses a grid environment. There are two ways of representing a grid-based environment. One is a X-Y coordinates plane [15] and the other is an orderly numbered grid, which has been widely used. We adopt the latter approach, in which a square environment is evenly divided into a certain number of squares, i.e., the x-axis and y-axis are divided equally into m parts, thus, we get $m \times m$ grids, where one or more grids are used to represent the obstacles. An example of the 7×7 grids is shown in Fig. 1(a), where the grid map is encoded by using Matlab and the grey shadow grids represent obstacles.

The mapping relations between coordinates (x, y) and the serial number p beginning from one can be identified by the following formula:

$$p = fix(y/SoG) \cdot NoC + fix(x/SoG) \tag{5}$$

where NoC is the number of columns; SoG is the size of a grid; the function $fix(t)$ rounds t to its nearest integer towards zero.

2.2 Related work

Since the pioneering work of Lozano-Pérez [12], a number of algorithms for solving the path planning problems have been reported in the past thirty years. These can be generally divided into two main classes: classical [14] and heuristic [19] algorithms. Although the classic approaches can be used to solve this problem, they may suffer from some drawbacks, such as easily falling into local minima, high complexity in high dimensions. In order to overcome these problems of classic methods, heuristic algorithms have been developed.

The representative heuristic approaches for solving MR3P are neural networks, genetic algorithms [15], ant colony optimization, fuzzy logic [16], simulated annealing [17], PSO [21], probabilistic road maps, rapidly exploring random trees, etc. Although heuristic methods do not guarantee to find an optimal solution, they may be faster and may have higher efficiency than classical methods [19]. The studies in [20, 21] have shown that the interest in PSO-based meta heuristics algorithms is growing in mobile robotics. Particularly interesting is the work in [20, 21] for solving the static or dynamic MR3P. According to the reports in the literature, the dimensions of the search space are set to a fixed value and remain constant throughout the entire optimization process in almost all of PSO-based algorithms for solving MR3P; consequently, the solving ability of the algorithms is limited to a single individual's dimension and the algorithm cannot find the optimal solutions. In MR3P, the dimensions of the search space decide the number of nodes of the optimal path. High dimensions may result in the decrease of the searching efficiency, while low dimensions may cause the case in which it is impossible to

get barrier-free paths. In order to find a proper dimension for a path and improve the search efficiency to self-adapt the dynamic environment with randomly appearing or disappearing obstacles, mMPSO with variable dimensions is introduced to solve MR3P and will be presented in the next section.

3 mMPSO for MR3P

This study considers a grid-based environment, in which obstacles or dangerous objects may appear or disappear. In order to describe the hostile environment, mMPSO uses a variable dimension PSO and a dynamic membrane structure with membrane division and dissolution. To improve the mMPSO performance such as effectiveness, efficiency and extensibility, we introduce a point repair algorithm, a smoothness approach and a moving direction adjustment technique. In what follows, we first present the variable dimensions and then describe the point repair algorithm and the smoothness approach. Finally, we summarize the mMPSO algorithm.

3.1 Variable Dimensions

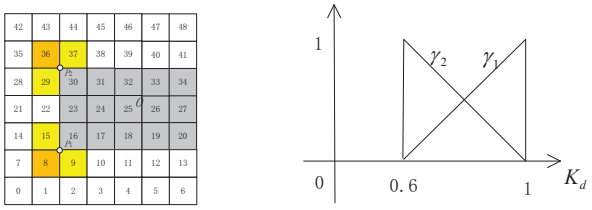
In mMPSO, each particle represents a feasible path, instead of an infeasible path in other heuristic approaches such as genetic algorithms. If the dimension of each particle is fixed, the search efficiency is often low due to the following reasons: (1) Population initialization, i.e., obtaining a population of initially feasible paths through randomly searching each node row by row, is time-consuming, especially for large grids or complex environments with circuitous route phenomenon. (2) The search process of the algorithm with fixed dimensions is also time-consuming, as compared with the dimension-reducing methods, because the variable dimensions in this study consider the removal of redundant information at each iteration. (3) Due to a complex environment, the optimization algorithms with fixed dimensions have low efficiency and poor adaptability.

To overcome these shortcomings, a set of high-dimension particles are needed at the beginning and the dimension of the best solution, i.e., the optimal path, is usually quite low. Thus, the dimensions of each particle in mMPSO are considered to be variable. In mMPSO for solving MR3P, the initial population P of particles (feasible paths) is classified into several subpopulations P_{min},...,P_{max}, where P_{min} is the subpopulation with lower dimensions, which represent shorter paths, that pass around fewer obstacles, and P_{max} is the subpopulation with higher dimensions, which denote longer paths, passing around more obstacles. At the beginning, the population size S_{min} of P_{min} may be similar to the one S_{max} of P_{max}, and the particles in P_{max} may search a feasible path through passing around external obstacles, while the particles in P_{min} may go to the contrary case. As the algorithm goes forward, S_{min} will increase and S_{max} will decrease. In general, a particle with low dimensions produces a shorter path, while a particle with high dimensions corresponds to a longer path. However, there are still some exceptions. But in mMPSO, the point repair algorithm, the smoothness approach and the moving direction adjustment technique can rectify the exceptions. The implementation of variable dimensions motivates the dynamic membrane structure of mMPSO.

3.2 Point Repair Algorithm

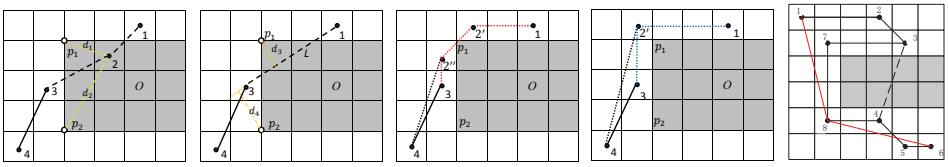
In the process of searching the optimal path, some nodes may move into obstacles and some path segments may cross obstacles, which results in infeasible paths and it is necessary to repair them. This study introduces a point repair algorithm to change the infeasible paths into feasible paths. We first define some special cells in the environment model. In Fig. 1(a), p_1, which is

surrounded by the three peripheral cells, 8, 9 and 15, and p_2, which is surrounded by the three peripheral cells, 29, 36 and 37, are the vertexes of obstacle O. All the peripheral cells have two kinds of coefficients, γ_1 and γ_2, which are randomly selected and are controlled by the weighting factors, K_d, K_s and K_f in (1). The coefficient γ_1 related to lateral cells $\{9, 15, 29, 37\}$ is mainly controlled by K_d. The coefficient γ_2 related to the diagonal cells $\{8, 36\}$ is mainly controlled by K_s and K_f, where $K_d + K_s + K_f = 1$, $0.6 \leq K_d \leq 1$, $0 \leq \gamma_1 \leq 1$, $0 \leq \gamma_2 \leq 1$, $\gamma_1+\gamma_2 = 1$. The relationship between γ_1 and γ_2 is shown in Fig. 1(b). For example, if $K_d = 0.8$, $K_s = K_f = 0.1$, we obtain $\gamma_1 = 0.5$ and $\gamma_2 = 0.5$. If $K_d = 1$, $K_s = K_f = 0$, we get $\gamma_1 = 1$ and $\gamma_2 = 0$.



(a) (b)

Figure 1: Definition of grids



(a) (b) (c) (d) (e)

Figure 2: An example for point repair and smoothness algorithms

Two types of infeasible paths are shown in Fig. 2(a)-(b), where d_1 is the point to point distance between node 2 and p_1; d_2 is the point to point distance between node 2 and p_2; d_3 is the point to line distance between p_1 and L; d_4 is the point to line distance between p_2 and L; d_1, d_2, d_3 and d_4 decide which peripheral cells will be selected. In Fig. 2(a), the node 2 in the path $\{1, 2, 3, 4\}$ (Type 1) must be repaired. In Fig. 2(b), the path segment L crossing the obstacle (Type 2) must be broken. The point repair process is as follows.

Step 1: Evaluate a path found. If it is feasible, we skip the repair process, otherwise, we perform the repair process.

Step 2: Judge the type of an infeasible path, Types 1 or 2.

Step 3: If the infeasible path is Type 2, go to Step 4, otherwise, conduct the following steps:

(a) Calculate the distance between the infeasible point p_0 and the vertex p_i of the obstacle, then get the value(s) of d_i, $i = 1$ (the obstacle is in the corner) or $i = 1, 2$ (the obstacle is located at the edge of the map) or $i = 1, 2, 3, 4$ (the obstacle is located in the middle of the map). Next, sort d_i in an increasing order.

(b) Select unused p_i by using the corresponding smallest value in d_i and get the peripheral cells $p^j_{g\_i}, j = 1, 2, 3$, and randomly select unused $p^j_{g\_i}$ by using γ_1 and γ_2.

(c) Replace the value of the infeasible node with selected $p^j_{g\_i}$.

(d) Judge the path again. If it is feasible, this process terminates, otherwise, go to step 2.

Step 4: Type 2 is repaired according to the following steps:

(a) Calculate the distance between the infeasible path segment L and the vertex p_i of the obstacle, get the value(s) of $d_i^{'}$, $i = 1$ (the obstacle is in the corner) or $i = 1, 2$ (the obstacle is located at the edge of the map) or $i = 1, 2, 3, 4$ (the obstacle is located in the middle of the map). Next, sort $d_i^{'}$ in an increasing order.

(b) Select unused p_i by using the corresponding smallest value in $d_i^{'}$ and get the peripheral cells $p_{g\_i}^j, j = 1, 2, 3$, and randomly select unused $p_{g\_i}^j$ by using γ_1 and γ_2.

(c) Insert the selected $p_{g\_i}^j$ between the two nodes of the infeasible path segment and get two new path segments $path_{s\_1}$ and $path_{s\_2}$.

(d) Judge each of the two paths $path_{s\_1}$ and $path_{s\_2}$. If one of them is not feasible, go to step 4, otherwise, the repair process terminates.

There are three cases of infeasible paths shown in Fig. 2(e) (dash line) and Fig. 2(a)-(b). We use the introduced point repair algorithm to process the three cases and obtain the corresponding results shown in Fig. 2(e) (solid line) and Fig. 2(c)-(d), respectively. E.g., the path segment $\{3, 4\}$ across the obstacle O in the infeasible path represented by the nodes $\{1, 2, 3, 4, 5, 6\}$ in Fig. 2(e) should be modified. If $\gamma_1 \gg \gamma_2$, the path segment $\{3, 4\}$ is replaced by two path segments $\{3, 8\}$ and $\{8, 4\}$ in the first modification, but the modified path segment $\{3, 8\}$ is still infeasible and must be modified further. In the second modification, the segment $\{3, 8\}$ is replaced by the path segments $\{3, 7\}$ and $\{7, 8\}$. Thus, all the path segments are feasible and the new path is $\{1, 2, 3, 7, 8, 4, 5, 6\}$. In Fig. 2(c)-(d), the new feasible path is $\left\{1, 2^{'}, 2^{''}, 3, 4\right\}$ or $\left\{1, 2^{'}, 3, 4\right\}$ under the condition $\gamma_1 = 0, \gamma_2 = 1$ or $\gamma_1 = 1, \gamma_2 = 0$.

3.3 Smoothness Algorithm

The smoothness algorithm is used to get rid of those redundant nodes of a feasible path. The smoothness process is described as follows:

Step 1: Sort the nodes in a path from the starting node to the goal node and get a sequence $n_i, i = 1, 2, \ldots, m$, where m is the dimension of a particle; n_1 and n_m are the starting and goal nodes, respectively.

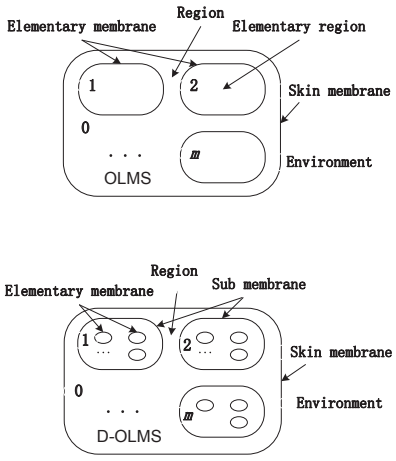
Step 2: Judge the path segment L_{ij} between n_i and n_j (at the beginning, $i = 1, j = 3$), if L_{ij} is infeasible, insert the nodes i and $j - 1$ into the node set P_f of the smoothed path, i.e., $P_f = \{i, j - 1\}$ and let $i = j - 1$ and j increases by 1, the algorithm continues to judge the path segment L_{ij}, otherwise, let j increases by 1, continue to judge the feasibility of the path segment L_{ij} till it is infeasible, insert the nodes i and $j - 1$ into the node set P_f of the smoothed path and let $i = j - 1$. Repeat this step till $j = m$.

As shown in Fig. 2(e), we use the introduced smoothness algorithm to remove the redundant nodes in the path $\{1, 2, 3, 7, 8, 4, 5, 6\}$ and obtain the smoothed path $\{1, 8, 6\}$. Similarly, the smoothed paths $\left\{1, 2^{'}, 2^{''}, 4\right\}$ and $\left\{1, 2^{'}, 4\right\}$ come from the paths $\left\{1, 2^{'}, 2^{''}, 3, 4\right\}$ and $\left\{1, 2^{'}, 3, 4\right\}$, respectively, as shown in Fig. 2(c)-(d).

3.4 mMPSO

In mMPSO, a dynamic membrane structure (specifically, OLMS alternates with D-OLMS shown in Fig. 3(a)) is introduced to arrange a population of particles, each of which is a feasible path for a mobile robot, and specify various rules, such as membrane division, transformation and communication-like rules, and membrane dissolution. The dimension of each particle is variable in the process of evolution. The point repair algorithm described above is used to change infeasible paths into feasible ones. The repair process may increase the dimensions of each particle. The smoothness algorithm presented in this section is applied to remove the redundant nodes of a path and the process may decrease the dimensions of each particle. In

addition, a moving direction adjustment technique is presented to accelerate the algorithm convergence. The pseudocode algorithm of mMPSO is shown in Fig. 3(b), where each step is described in detail as follows:



(a)

Begin:
$t \leftarrow 1$
(1) Initialize membrane structure
(2) Initialize populations
(3) Produce new particles in each elementary membrane
(4) **While** (not termination condition) **do**
(5) Divide each elementary membrane
(6) Evaluate every particle
(7) Find $P_{best}^{id}(t)$
(8) Find local best particle $G_{ij}^{b}(t)$
(9) Execute communication rules (a)
(10) Find global best particle $G_{d}^{b}(t)$
(11) Execute communication rules (b)
(12) Update particle's velocity $V(t)$
(13) Update particle's position $X(t)$
(14) Execute point repair algorithm
(15) Execute smoothness algorithm
(16) Adjust each particle's moving direction
(17) Dissolve elementary membrane
$t \leftarrow t+1$
End
End

(b)

Figure 3: Membrane structures and the pseudocode algorithm for mMPSO

Step 1: An OLMS $[[\,]_1, \ldots, [\,]_m]_0$ composed of a skin membrane denoted by 0 and $m+1$ regions inside the skin membrane is constructed. The way in which the parameter m is chosen will be discussed in Section 4.

Step 2: A particle swarm X with m particles in a D-dimensional search space is randomly generated and each particle is put inside an elementary membrane in OLMS, where D represents the number of nodes in a feasible path; $X = \{x_1, x_2, \ldots, x_m\}$, where x_i is an arbitrary individual in X and denotes a feasible path, $x_i = (x_{i1}, x_{i2}, \ldots, x_{iD})$.

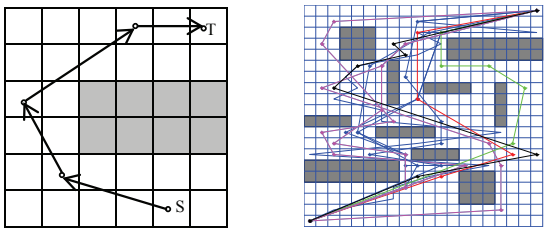
Step 3: In this step, a moving direction adjustment technique is introduced to produce n particles inside each elementary membrane. To be specific, we modify the velocity of the particle inside each elementary membrane to generate a new particle by using (6) and (7),

$$V(g+1) = \rho_1 \cdot V_r(g) + \rho_2 \cdot V_f(g) \tag{6}$$

$$X(g+1) = X(g) + V(g+1) \tag{7}$$

where ρ_1 and ρ_2 are the inertia weighting factors and usually are set to larger values for exploring the global solutions; $V_r(g)$ is the randomly produced velocity of the gth particle (at the beginning for each elementary membrane, g=0); $V_f(g)$ is the adjusted velocity of the gth particle by using the idea of tightening fishing line and the moving directions of each node in the gth particle is shown in Fig. 4(a); $V(g+1)$ is the velocity of the $(g+1)$th particle; $X(g)$ and $X(g+1)$ are positions of the gth and $(g+1)$th particles. Inspired by the idea of tightening fishing line, we consider a feasible path as a fishing line and tighten the line from the target node, thus, each node except for the target one in the path will show a moving direction toward the next node (the target node is the first one). The moving directions of all the nodes in the path construct the velocity $V_f(g)$. This step is repeated for n times to produce n new particles for each elementary membrane and used together with the point repair algorithm and smoothness algorithm. The dimensions of new particles may be greater than or less than D. Thus, the swarm will have

$m \times n$ particles in total. Fig. 4(b) shows an example that one particle with 20 dimensions (the thick line) inside a certain elementary membrane produces 10 particles with 6–10 dimensions (the thin line). Compared to the random approach, the production of the new particles can remove redundant nodes of a path and has better adaptability in hostile environment, especially in the circuitous route environment.



(a) (b)

Figure 4: An example of the generation of new particles and direction of each dimension of the individual

Step 4: The maximal number of iterations is used as the termination condition.

Step 5: This step first classifies the n particles inside the ith elementary membrane into k_i clusters according to the dimension of each particle and then divides the ith elementary membrane into k_i membranes, each of which contains the particles with the same dimension, $i = 1, 2, \ldots, m$. Thus, OLMS becomes D-OLMS.

Step 6: Each particle is evaluated by using (1) and assigned a fitness value.

Step 7: Find $P_{best}^{id}(t)$, which is the best solution of each particle in its history with respect to the fitness values.

Step 8: Find the locally best solution $G_{ij}^{b}(t)$ in the jth elementary membrane inside the ith membrane, $i = 1, 2, \ldots, m, j = 1, 2 \ldots, k_i$, in terms of fitness values.

Step 9: Perform communication rules (a), which first send all the locally best solutions $G_{ij}^{b}(t)$ ($j = 1, 2 \ldots, k_i$) out into the ith submembrane, $i = 1, 2, \ldots, m$, and further send out into the skin membrane.

Step 10: Find the globally best solution $G_{d}^{b}(t)$ by comparing $G_{ij}^{b}(t)$ with the same dimension d, $d \in \{1, 2, \ldots, D\}$, $i = 1, 2, \ldots, m, j = 1, 2 \ldots, k_i$.

Step 11: Perform communication rules (b), which send $G_{d}^{b}(t)$ back into the elementary membrane containing d-dimension particles across certain submembrane.

Step 12: Update the velocities of the d-dimension particles using (8).

$$\begin{aligned} V_{id}(t+1) = \delta_1 \cdot \Big(& \rho \cdot V_{id}(t) + c_1 \cdot r_1 \cdot \big(P_{best}^{id}(t) - X_{id}(t)\big) + c_2 \cdot r_2 \cdot \Big(G_{ij}^{b}(t) - X_{id}(t)\Big) \\ & + c_3 \cdot r_3 \cdot \big(G_{d}^{b}(t) - X_{id}(t)\big)\Big) + \delta_2 \cdot V_{id}^{f}(t) \end{aligned} \tag{8}$$

where $V_{id}(t)$ and $V_{id}(t+1)$ are the velocities of the particle at generation t and $t+1$, respectively; $P_{best}^{id}(t)$ is the best solution of the particle at generation t; $X_{id}(t)$ is the position the particle at generation t; $G_{ij}^{b}(t)$ is the locally best solution with the same dimension d at generation t; $G_{d}^{b}(t)$ is the globally best solution with the same dimension d at generation t; $V_{id}^{f}(t)$ is the adjusted velocity of the particle at generation t; δ_1 and δ_2 are proportion coefficients; ρ is an inertia weighting factor; r_1, r_2 and r_3 represent the functions that generate independently random numbers, which are uniformly distributed between 0 and 1; c_1, c_2 and c_3 are acceleration coefficients.

Step 13: Update the positions of the d-dimension particles using (9).

$$X_{id}(t+1) = X_{id}(t) + V_{id}(t+1) \tag{9}$$

where $X_{id}(t)$ and $X_{id}(t+1)$ are the positions of the particle at generation t and t+1, respectively; $V_{id}(t+1)$ is the velocity of the particle at generation $t+1$.

Step 14: Execute point repair algorithm for each particle.

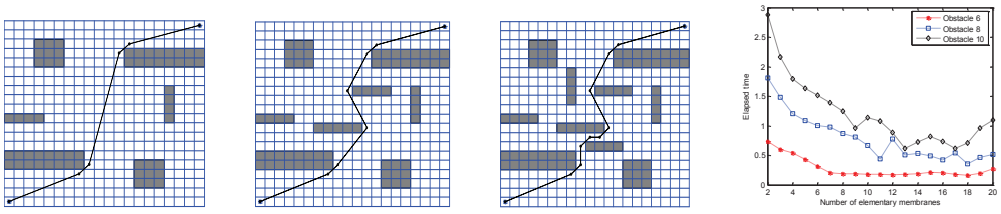
Step 15: Execute smoothness algorithm for each particle.

Step 16: Adjust the moving direction of each particle by using the moving direction adjustment technique.

Step 17: This step dissolves all the elementary membranes and releases their particles into their corresponding submembranes. Thus, D-OLMS becomes the original OLMS.

4 Experimental Results

The mMPSO performance is tested by using MR3P. We first discuss how to set the number m of elementary membranes in OLMS by using 20×20 grid model environment with 6, 8 and 10 obstacles, respectively. Then, 16×16 grid model environment with 9 static obstacles are applied to compare mMPSO with its counterpart vPSO and GA [15]. Subsequently, the complex environments, 32×32 and 64×64 grid model environments with 20 static obstacles, are applied to further test the mMPSO performance. In these experiments, one dynamic obstacle representing a moving obstacle or a dangerous source occurring suddenly is employed to analyze the mMPSO behavior.



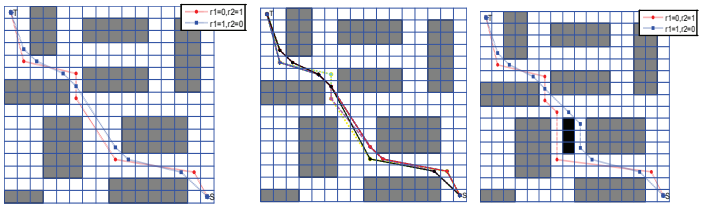
(a) 6 obstacles (b) 8 obstacles (c) 10 obstacles (d) Results

Figure 5: The near shortest paths in three environments (20×20 grid with obstacles 6, 8 and 10, respectively) and experiment results

4.1 Parameter Setting

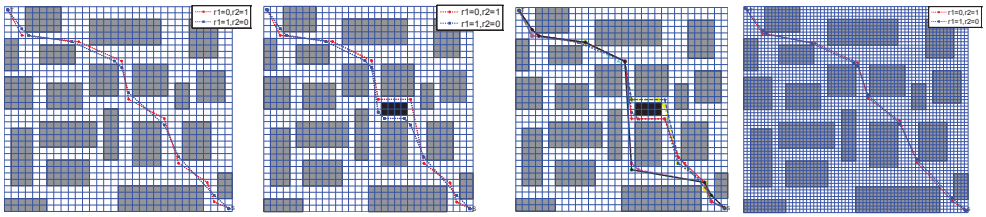
In this subsection, we use 20×20 grid model environment with three kinds of obstacles shown in Fig. 5 to discuss how to choose m. Fig. 5(a)-(c) have 6, 8, 10 static obstacles (shaded areas), respectively. In the following experiments, the population size is set to 100; In (8), $c_1 = c_2 = c_3 = G_{size}/V_{\max}$, where $G_{size} = 0.08$ is the size of a grid and $V_{\max}$ is the maximal distance allowing a node to move in a step; the proportion coefficients $\delta_1 = 0.65$, $\delta_2 = 0.35$; ρ is defined as a variable, which varies from 0.246 to 0.157 along the logarithm function $\log_{10}(y)$. In (6), ρ_1 and ρ_2 are set to random values between 0.4 and 0.6. In (3), $\alpha = 0.6$, $\beta = 0.4$. In (4), λ is set to the robot radius.

In what follows, m varies from 2 to 20 by an increment of 2, thus, we first generate m particles in *Step 2* and in *Step 3* for the first $m-1$ elementary membranes, we produce $round(100/m)$ particles and $100-(m-1)*round(100/m)$ particles for the mth elementary membrane, where $round(.)$ is a function for rounding its element towards nearest integer. In the experiments, if a given near-optimal solution is reached, mMPSO stops. Because the optimal solution to MR3P is usually unknown, we set $K_d = 1, K_s = K_f = 0$ in (1) and independently perform mMPSO for 30 times, where the terminal condition is such that the maximal number of iterations is



(a) (b) (c)

Figure 6: Experimental results of mMPSO in the environments 16×16 grids, $O_s = 9$ and $O_d = 1$.



(a) 32×32 (b) 32×32 (c) 32×32 (d) 64×64

Figure 7: Experimental results of mMPSO in the environments 32×32, 64×64 grids, $O_s = 20$ and $O_d = 1$.

set to 2000, in order to find the near-optimal solution. Fig. 5(a)-(c) show the near shortest paths of the model environment with different obstacles, 6, 8 and 10, respectively. The mMPSO performance for each of the 19 cases is evaluated by using the mean of the elapsed time in 30 independent runs. The experimental results are shown in Fig. 5(d), where the elapsed time for three environments first decreases and then increases as the value of m goes up. These experimental results indicate that m could be 13 by considering the three environments.

4.2 MR3P Experimental Results

To investigate the mMPSO performance, this subsection uses three grid models, 16×16, 32×32 and 64×64, to carry out the experiments and considers five environments: 16×16 with 9 static obstacles ($O_s = 9$), 16×16 with $O_s = 9$ and one dynamic obstacle ($O_d = 1$), 32×32 with $O_s = 20$, 32×32 with $O_s = 20$ and $O_d = 1$, 64×64 with $O_s = 20$. The place for the possible occurrence of the dynamic obstacle is set to the near center, which is very likely to block the feasible paths. The model with 16×16 grids is applied to compare mMPSO with its counterpart PSO (vPSO) and GA in [15]. The models with 32×32 and 64×64 grids are used to further discuss the mMPSO performance in different complex environments. The setting of the parameters in mMPSO except for K_d, K_s, K_f is the same as in Subsection 4.1. m=13. The termination condition is designated as the maximal number 2000 of iterations. All the experiments are run on the PC with CPU 1.7GHz, 512MB RAM, and the software platform MATLAB7.4 and Windows XP OS.

We first use the model with 16×16 grids to compare mMPSO with vPSO (when $m = 1$, mMPSO becomes vPSO) and GA in [15]. We consider three cases for K_d, K_s, K_f as follows:

(1) Case 1: $K_d = 1, K_s = K_f = 0, \gamma_1 = 1, \gamma_2 = 0$;

(2) Case 2: $K_d = 0.6, K_s = K_f = 0.2, \gamma_1 = 0, \gamma_2 = 1$;

(3) Case 3: $K_d = 0.8, K_s + K_f = 0.2, \gamma_1 + \gamma_2 = 1$.

The experimental results of mMPSO are shown in Fig. 6. In Fig. 6(a), the blue line is the

Method	NoO	NoNO	NoI	Fv	Gn	St
GA[36]	9	78	13	24.68	16	1.68
vPSO	83	108	0	24.95	65	2.97
mMPSO	94	239	0	24.26	27	0.84

Table 1: Comparisons of three methods in the environment(Fig. 6 (a))

Method	NoO	NoNO	NoI	Fv	Gn	St
GA[36]	32	68	0	24.71	12	0.69
vPSO	81	103	0	28.56	73	3.12
mMPSO	92	235	0	27.43	34	0.97

Table 2: Comparisons of three methods in the environment(Fig. 6 (c))

best path in Case 1 considering only one objective, path length, and the red line is the best result in Case 2 by trading-off safety and smoothness. Fig. 6(b) illustrates 8 near optimal paths (8 colors) through balancing the path length, safety and smoothness. The paths in Fig. 6(c) are obtained by considering one dynamic obstacle and the blue line is the best path in Case 1 considering only one objective, path length, and the red line is the best result in Case 2 by trading-off safety and smoothness.

To draw a comparison with GA in [15] and vPSO, let K_d=1 and the experiment is executed for 100 independent runs. Tables 1 and 2 show the experimental results of GA, vPSO and mMPSO for the environments with static obstacles and the environments with static and dynamic obstacles. In Tables 1-3 , NoO, NoNO, NoI, Fv, Gn, St represents the number of optimal solutions, the number of near optimal solutions, the number of infeasible solutions and the fitness value in 100 trials, the average generations for finding the optimal solution and the mean of the elapsed time(s) in each trial, respectively.

As it can be clearly seen from Table 1 and Fig. 6, mMPSO finds much more optimal paths and near optimal paths, while it spends smaller computing time than GA. There are some infeasible solutions in GA, while there is not any infeasible solution in vPSO and mMPSO because the point repair algorithm have repaired the infeasible path. On the other hand, vPSO also finds more optimal paths and near optimal solutions than GA, but the elapsed time is far larger than GA. mMPSO is better than vPSO with respect to optimal and near optimal solutions and the elapsed time, which indicates the advantage of the combination of a membrane system with PSO. Table 2 shows similar conclusions to those in Table 1.

To further analyze the mMPSO performance in more complex environments, more experiments are conducted in the environments with 32×32 and 64×64 grids containing 20 or 21 obstacles, as shown in Fig. 7 (a-d). The environment with 32×32 grids and 20 static obstacles

Environment	NoO	NoNo	Fv	Gn	St
32×32, $O_s = 20$, $O_d = 0$	86	242	28.79	36	1.72
32×32, $O_s = 20$, $O_d = 1$	82	225	31.53	45	1.93
64×64,$O_s = 20$, $O_d = 0$	83	247	28.14	59	2.68

Table 3: Experimental results of mMPSO in different environments in Fig. 7

are shown in Fig. 7 (a). Fig. 7(b)-(c) show the environment with 20 static obstacles and one dynamic obstacle. In Fig. 7 (c), the three objectives, path length, smoothness and safety, are considered. The parameters of mMPSO are the same as above except for the population size 150 and $m = 15$. All the tests are executed for 100 independent runs. Table 3 shows the results.

It can be seen from Tables 1-3 that the optimal solutions of mMPSO drop from 94 to 83, the elapsed time rises from 0.84 to 2.68 and the average generations vary from 27 to 59 as the number of cells increases from 16×16 to 64×64 and the static obstacles go up from 9 to 20. The elapsed time and average generations increase a little with the dynamic obstacle. To sum up, as the number of cells increases in accordance with 4^n $(n = 1, 2, 3 \ldots)$ values and the static obstacles double, the increase in the elapsed time is quite small, as opposed to an exponential growth in cells number. mMPSO maintains good search capability to find the optimal solution in both static and dynamic environments, which indicates mMPSO has good adaptability to MR3P under complex environments.

5 Conclusions

This paper discusses a feasible combination of membrane systems and PSO to solve MR3P. The outstanding novelty is to justify the introduced dynamic membrane structure, which proves to be suitable for solving MR3P with variable dimensions. mMPSO uses the alternation of OLMS and D-OLMS to integrate a PSO with variable dimensions, point repair algorithm, smoothness algorithm and moving direction adjustment. A large number of experiments are carried out on several MR3P with various environments and the results show that mMPSO can achieve much better solutions than its counterparts PSO and GA, as reported in the literature.

This paper considers only the planar (two dimensions) environments. Following this work, we aim to further investigate how to extend mMPSO to three dimensional spaces, how to use mMPSO to solve more difficult path planning problems (mobile robots follow the tracks of moving targets in a hostile environments), how to combine mMPSO with numerical P systems to control mobile robots and how to apply the idea of variable dimension PSO to solve other engineering problems.

Acknowledgment

The work of XW, GZ, JZ and HR was supported by the National Natural Science Foundation of China (61170016, 61373047), the Program for New Century Excellent Talents in University (NCET-11-0715) and SWJTU supported project (SWJTU12CX008). The work of FI and RL was supported by a grant of the Romanian National Authority for Scientific Research, CNCS-UEFISCDI, project number PN-II-ID-PCE- 2011-3-0688.

References

[1] Păun, Gh.; Rozenberg, G.; Salomaa, A. eds,(2010); The Oxford handbook of membrane computing, Oxford University Press.

[2] Păun, Gh. (2007); Tracing some open problems in membrane computing, *Rom J Inform Sci Tech*, ISSN 1453-8245, 10(4): 303–314.

[3] Zhang, G.X.; Pan, L.Q. (2010); A survey of membrane computing as a new branch of natural computing, *Chinese J Computer*, ISSN 0254-4164, 33(2): 208–214.

[4] Nishida T.Y. (2004); An application of P system: a new algorithm for NP-complete optimization problems. *Proc 8th WMCSCI*, 109–112.

[5] Zhang, G.X.; Gheorghe, M.; Pan, L.Q.; Pérez-Jiménez, M.J. (2014); Evolutionary membrane computing: a comprehensive survey and new results, *Inform Sci* (accepted).

[6] Huang, L.; He, X.; Wang, N.; Xie, Y. (2007); P systems based multi-objective optimization algorithm, *Prog Nat Sci*, ISSN 1002–0071, 17(4): 458-465.

[7] Zhang, G.X.; Gheorghe, M.; Li, Y. (2012); A membrane algorithm with quantum-inspired subalgorithms and its application to image processing, *Nat Comput*, ISSN 1567-7818, 11(3): 701–717.

[8] Zhang, G.X.; Cheng, J.X; Gheorghe, M. (2014); Dynamic behavior analysis of membrane-inspired evolutionary algorithms, *Int J Comput Commun Control*, ISSN 1841-9836, 9(2): 227–242.

[9] Kennedy, J.; Eberhart, R. (1995); Particle swarm optimization, *Proc ICNN*, 4: 1942–1948.

[10] Zhang, G.X.; Zhou, F.; Huang, X.L; Cheng, J.X; Gheorghe, M; Ipate, F.; Lefticaru, R. (2012); A novel membrane algorithm based on particle swarm optimization for solving broadcasting problems, *J Univers Comput Sci*, ISSN 0948-6968 18(13): 1821–1841.

[11] Xiao, J.; Huang, Y.; Cheng, Z; He, J; Niu, Y. (2014); A hybrid membrane evolutionary algorithm for solving constrained optimization problems, *Optik*, ISSN 0030-4026, 125(2): 897–902.

[12] Lozano-Pérez, T.; Wesley, M.A. (1979); An algorithm for planning collision-free paths among polyhedral obstacles, *Commun ACM*, ISSN 0001-0782, 22(10): 560–570.

[13] Gao, H.; Cao, J. (2012); Membrane quantum particle swarm optimisation for cognitive radio spectrum allocation, *Int J Comput Appl Tech*, ISSN 0952-8091, 43(4): 359–365.

[14] Hwang, Y.K.; Ahuja, N. (1992); Gross motion planning - A survey, *ACM Comp Surv*, 24: 219-291.

[15] Tuncer, A.; Yildirim, M. (2012); Dynamic path planning of mobile robots with improved genetic algorithm, *Comput Electr Eng*, ISSN 0045-7906, 38: 1564–1572.

[16] Garcia, M.A.; Montiel, O. (2009); Path planning for autonomous mobile robot navigation with ant colony optimization and fuzzy cost function evaluation, *Appl Soft Comput*, ISSN 1568-4946, 9(3): 1102–1110.

[17] Qidan, Z.; Yang, Y.J.; Xing, Z.Y. (2006); Robot path planning based on artificial potential field approach with simulated annealing, *Proc ISDA*, 622–627.

[18] Zhang, Y.; Gong, D.W. (2013); Robot path planning in uncertain environment using multi-objective particle swarm optimization, *Neurocomputing*, ISSN 0925-2312, 103: 172–185.

[19] Masehian, E.; Sedighizadeh, D. (2007); Classic and heuristic approaches in robot motion planning - A chronological review, *Proc. WASET*, 101–106.

[20] Xu, W.F.; Li, C.; Liang, B. (2008); The cartesian path planning of free-floating space robot using particle swarm optimization, *Int J Adv Rob Syst*, ISSN 1729-8806, 5: 301–310.

[21] Gong, D.W.; Zhang, J.H. (2011); Multi-objective particle swarm optimization for robot path planning in environment with danger sources, *J Comput*, 6(8): 1554–1561.