Generation of rapidly-exploring random trees by using a new class of membrane systems

Ignacio Pérez-Hurtado, Mario J. Pérez-Jiménez

Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
Universidad de Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: {perezh,marper}@us.es

Abstract. Methods based on Rapidly-exploring Random Trees (RRTs) have been in use in robotics to solve motion planning problems for nearly two decades. On the other hand, models based on Enzymatic Numerical P systems (ENPS) have been applied to robot controllers for more than six years. These controllers in real robots handle the power of motors according to motion commands usually generated by planning algorithms, but today there is a lack of planning algorithms based on membrane systems for robotics. With this motivation, we provide in this paper a new variant of ENPS called Random Enzymatic Numerical P systems with Proteins and Shared Memory (RENPSM) oriented to RRTs for planning in robotics and we illustrate it by presenting a model for generation of RRTs with holonomic limitations. We are working on the ENPS framework with the idea of moving towards a complete mobile robot system based on membrane systems, i.e. including controllers and planning; and we have incorporated new ingredients into the ENPS framework to meet the requirements of the RRT generation algorithm.

1 Introduction

Robots are machines oriented to objectives equipped with actuators, sensors and computation units acting under physical constraints. Regardless of their morphology, they should accomplish tasks by acting in the real world. This is one of the main reasons by which Robot Motion Planning [4] is an eminent research area in Robotics. In general terms, the problem of motion planning can be defined in the configuration space of a robot as follows: *Given (1) a start configuration state, (2) a goal configuration state, (3) a geometric description of the robot, and (4) a geometric description of the environment: find a path that moves the robot gradually from start to goal.*

A configuration state is a specification of the positions of all robot points relative to a fixed coordinate system. This is usually expressed as a vector of positions and orientations, for example, a rigid-body robot in a 2D world can be expressed as a vector (x, y, θ) representing the center (x, y) of the robot in a fixed coordinate system and its yaw angle θ. Since the shape of the robot is described, all of its points are then known.

Several constraints can be added to this problem, the most common is to reach the goal while never touching any obstacle in the environment. Others can also be added, for example, a social robot could restrict configuration states in order to guarantee the human comfort.

The configuration space of a robot can also be constrained by the type of movements the robot can perform. In this sense, non-holonomic robots are those that cannot instantly modify its direction without employing rotation in-place. On the other hand, holonomic robots can do it (assuming zero mass). For example, a holonomic robot in a 2D world can move along the x axis and the y axis, as well as modify its yaw angle if needed. But a non-holonomic robot can only move forward/backward and/or modify its yaw angle. This is the typical case of dual-wheeled mobile robots.

Classical path planning algorithms have been widely adapted and applied to the problem of motion planning with constraints in robots, for example, in [16], an application of the Dijkstra for robot path-planning was presented. In such solutions, the general problem is usually divided into two smaller problems: the *global path planning* problem, as described above; and the *local path planning*, where the robot tries to connect two consecutive states in real-time considering constraints not included in the global plan as, for example, obstacles in movement. The accumulated error during the local planning conducts to periodically recompute the global plan. For this reason, the computational complexity of global planners is a critical point regarding to real-time constraints. Many efforts have been made to provide good global planners. For example: in [15], a new search algorithm, called D^*, is presented for path planning in real-time environments. In [7], a variant of the classical search algorithm A^* is applied to grids with blocked and unblocked cells. In [5], a new tool for path planning, called Rapidly-Exploring Random Trees (RRT), was presented.

The class of RRT algorithms for path planning is based on the randomized exploration of the constrained configuration space by building a tree where nodes represent states that can be reached by the state of the corresponding parent in a fixed amount of time, so each edge contains a valid motion command to reach the state in child node. It is currently one popular method in Robot Motion Planning due to its good properties. The created RRT can be used to explore the constrained configuration space by applying search algorithms or, as presented in [6], the RRT generation algorithm can be used by itself as path planning algorithm, where two RRTs are built simultaneously, one beginning from the start configuration and another one beginning from the goal configuration.

In order to execute the planned motions, each motor of the robot must be able to maintain a certain speed command for a fixed period of time. This is the function of a type of software called *controller* on-board of the robot. Thus, *Robot Control* [1] is the branch of robotics dedicated to the study and practice of controlling robots.

Robot controllers are usually based on common silicon microprocessors, but in the recent years, some classes of *membrane systems* [8] have been in use for modelling them [12] [10] [11] [18]. Membrane systems (or P systems) are models

of computation based on the structure and functions of the living cells. In a Membrane system, there are objects being evolved inside compartments according to rules in a non-deterministic way with a high-level of parallelism. They were firstly used as a new technique to attack the P versus NP problem [13], but several applications of membrane systems have been also presented: Stochastic P systems for modelling biological phenomena [14], Probabilistic P systems for modelling real ecosystems [2], Spiking Neural P systems incorporating fuzzy reasoning, for fault diagnosis and learning [17], and others.

In relation to Robot Control, Numerical P Systems (NPS) were used for modelling and simulating robot controllers [12], although the initial application of NPS was related to models of Economical Processes [9]. Enzymatic Numerical P systems (ENPS) [10] were introduced as an improvement and applied to the distributed control of a swarm of mobile robots. Indeed, reactive and proportional-integral-derivative (PID) dual-wheeled robot controllers have been successfully designed and simulated by means of ENPS, as well as software simulator tools [18]. On the other hand, the robot must know its position in the environment by using sensors, that is the *Robot Localization* problem [3], and it has also been attacked by using ENPS [11].

The main contribution of this paper is to introduce a new variant of ENPS called *Random enzymatic numerical P systems with proteins and shared memory (RENPSM)*, in order to simulate RRT algorithms. Since the current applications of membrane systems to Robotics are focused on controllers, localization and local planners, we propose an approximation from "the other side": global path planning with physical constraints. The idea is to provide a complementary approach walking towards a mobile dual-wheeled robot system based completely on membrane systems. We are using ENPS because it is the framework of the state of the art in membrane computing and robotics. On the other hand, the motivation of the new ingredients is related to meet the requirements for the computation of the RRT algorithm:

- Random numbers: Since the RRT algorithm is a randomized method to explore the physical space.
- Shared memory: Because the algorithm can be parallelized using processes sharing common variables (for instance, at some instant each membrane must read a specific value stored in a distinguished membrane: the shared memory).
- Proteins: Since the algorithm performs steps that must be sequentially executed, proteins for synchronization are used.

This paper is structured as follows. In the next section, the rapidly-exploring random trees (RRTs) are described with some details. Section 3 is devoted to introduce the Membrane Computing framework called *random enzymatic numerical P systems with proteins and shared memory* (RENPSM, for short), where the RRT generation algorithm will be simulated. In Section 4, the simulation of the RRT algorithm by using RENPSM systems is presented. Finally, conclusions and future work are drawn.

2 Rapidly-exploring Random Trees

A RRT [5] is a randomized tree structure for rapidly exploring a *state space* X from an initial position/state x_{init}. It can be successfully used for nonholonomic and kinodynamic path planning in robotics [6].

Nodes in a RRT represent possible reachable states in a 2D or 3D world, for robots with nonholonimic constraints in a 2D environment this is given by (x, y, θ) where (x, y) are the Cartesian coordinates of the robot position and θ is the heading angle. If we consider a kinodynamic planning problem, a node encodes both position and velocity of the robot. For robots with holonomic constraints in a 2D world, the heading angle can be ignored.

It is assumed that a fixed *obstacle region* $X_{obs} \subset X$ must be avoided, so the nodes of the RRT are states in X_{free}, the complement of X_{obs} in X.

Edges in a RRT represent transitions between reachable states, they are labelled with the command u that the robot should execute for a fixed amount of time Δt in order to change the corresponding states. For a dual-wheeled robot with nonholonomic constraints, it can be represented by the pair of linear and angular velocity (v, ω) to be sent to the controller.

If we are working with holonomic constraints, all the points in the space can be reached applying a linear velocity, so edges are labelled with instant linear velocities and a new state x_{new} can be reached from another state x connected by an edge labelled with u by applying $x_{new} = x + u \cdot \Delta t$

Algorithm 1 GENERATE_RRT$(x_{init}, K, \rho, \Delta t, X, X_{obs}, d_{min})$

```
V_τ ← {x_init}
E_τ ← ∅
for k = 1 to K do
  x_rand ← RANDOM_STATE(X);
  x_near ← NEAREST_NEIGHBOR(x_rand, τ);
  if DISTANCE(x_rand, x_near) ≥ d_min then
    u ← SELECT_INPUT(x_rand, x_near);
    if ¬ COLLISION(x_near, u, Δt, X_obs) then
      x_new ← NEW_STATE(x_near, u, Δt);
      V_τ ← V_τ ∪ {x_new}
      E_τ ← E_τ ∪ {(x_near, x_new)}
    end if
  end if
end for
return τ = (V_τ, E_τ)
```

This is an iterative algorithm to generate a RRT [5], where:

- x_{init} is the initial state.
- K is the number of iterations to build the RRT.
- ρ is a prefixed distance metric.

- Δt is a fixed amount of time for transitions.
- X is the state space.
- X_{obs} is the obstacle state space.
- d_{min} is the minimum distance threshold according to ρ in order to include a new node in the RRT.
- $\tau = (V_\tau, E_\tau)$ is the RRT generated.
- RANDOM_STATE(X) is a function to get a random state from X
- NEAREST_NEIGHBOR(x_{rand}, τ) is a function to get the closest node to x_{rand} in τ according to ρ.
- DISTANCE(x_{rand}, x_{near}) is a function to get the distance of x_{rand} to x_{near} according to ρ.
- SELECT_INPUT(x_{rand}, x_{near}) is a function to get the velocity input that should be commanded to the robot in order to achieve state x_{rand} from x_{near}.
- COLLISION($x_{near}, u, \Delta t, X_{obs}$) is a function returning *true* if a collision could be produced moving the robot from state x_{near} by applying the input u for Δt time considering the obstacles in X_{obs}.
- NEW_STATE($x_{near}, u, \Delta t$) is a function to get a new state x_{new} by applying the input u to the robot for Δt time starting at state x_{near}.

In Figure 1 it is represented a RRT generated after 5000 iterations by using Algorithm 1 with holonomic constraints and the Euclidean distance as distance metric.

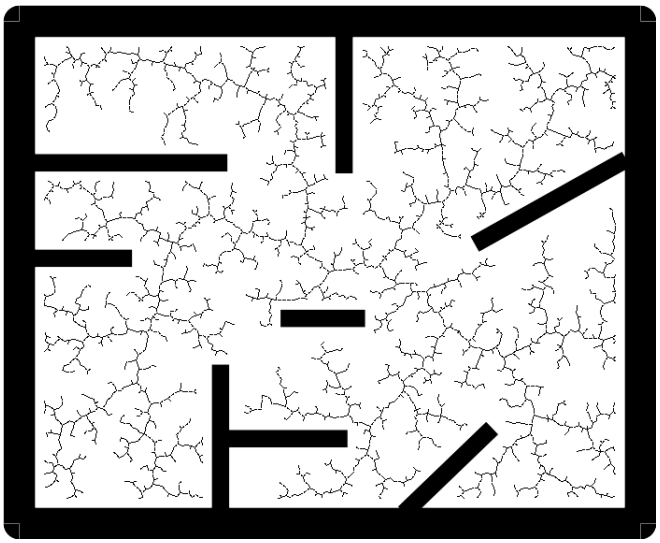

Fig. 1. RRT generated after 5000 iterations

In order to provide a first approximation based on Membrane Computing, we use in this paper a simplified version of the standard RRT algorithm 2 for

holonomic robots where no obstacles have been considered and ρ is the euclidean distance.

Algorithm 2 GENERATE_RRT$((x_0, y_0), K, \Delta t, X, d_{min})$

$V_\tau \leftarrow \{(x_0, y_0)\}$
$E_\tau \leftarrow \emptyset$
for $k = 1$ **to** K **do**
  $(x_{rand}, y_{rand}) \leftarrow$ RANDOM_STATE(X);
  $(x_{near}, y_{near}) \leftarrow$ NEAREST_NEIGHBOR$((x_{rand}, y_{rand}), \tau)$;
  $z \leftarrow \rho((x_{rand}, y_{rand}), (x_{near}, y_{near}))$
  if $z \geq d_{min}$ **then**
    $u_1 \leftarrow (x_{rand} - x_{near})/z$,
    $u_2 \leftarrow (y_{rand} - y_{near})/z$,
    $x_{new} \leftarrow x_{near} + u_1 \cdot \Delta t$
    $y_{new} \leftarrow y_{near} + u_2 \cdot \Delta t$
    $V_\tau \leftarrow V_\tau \cup \{(x_{new}, y_{new})\}$
    $E_\tau \leftarrow E_\tau \cup \{(x_{near}, y_{near}), (x_{new}, y_{new})\}$
  end if
end for
return $\tau = (V_\tau, E_\tau)$

3 Random enzymatic numerical P systems with proteins and shared memory

In this section a variant of *enzymatic numerical P systems* incorporating new features is presented in order to simulate the generation of RRTs.

Definition 1. *A random enzymatic numerical P systems with proteins and shared memory (RENPSM, for short) of degree* (p, q), $p, q \geq 1$, *is a tuple*

$$\Pi = (H, \mu, P, E_{mem}, E_{mem}(0), \{(P_h(0), Var_h, Var_h(0), Pr_h) \mid h \in H\}, \mathcal{R}, h_0)$$

where:

1. $H = \{1, \ldots, p \cdot q\} \cup \{mem\}$, $mem \notin \{1, \ldots, p \cdot q\}$, *is the set of labels of the system;*
2. μ *is a dynamical membrane structure (a rooted tree) initially consisting of only one* membrane *with label* $h_0 \in \{1, \ldots, p \cdot q\}$ *in such manner that along the computation new membranes will be created with labels in* $\{1, \ldots, p \cdot q\}$*;*
3. mem *is the label of a distinguished component (the shared memory of the system);*
4. P *is a finite set of objects, called catalyzator proteins, and* $P_h(0)$ *is the protein initially associated with region labeled by* h*;*

5. E_{mem} *is a finite set of variables, called enzymes, disjoint with* Var_{mem}*, and* $E_{mem}(0)$ *is the initial values of the enzymes;*
6. $Var_h, h \in H$*, is a finite set of variables* $x_{j,h}$ *associated with region (a membrane or the shared memory) labeled by* h*, its values must be natural numbers (they can also be zero), the value of* $x_{j,i}$ *at time* $t \in \mathbf{N}$ *is denoted by* $x_{j,i}(t)$*;*
7. $Var_h(0)$ *is a vector that represents the initial values for variables in* Var_h*;*
8. $Pr_h, h \in H$*, is a finite set of programs associated with region labeled by* h*, having the following syntactical format:*

$$F(x_{1,h}, \ldots, x_{k_F,h}) \xrightarrow{e(F);\alpha(F)} c_1|v_1, \ldots, c_{n_F}|v_{n_F}$$

where:
   - $F(x_{1,h}, \ldots, x_{k_F,h})$ *is a computable function (the production function), being* $x_{1,h}, \ldots, x_{k_F,h}$ *variables in* Var_h*;*
   - $c_1|v_1, \ldots, c_{n_F}|v_{n_F}$ *is the repartition protocol associated with the program, being* $c_1, \ldots, c_{n_F}$ *natural numbers specifying the proportion of the current production distributed to variables* $v_1, \ldots, v_{n_F} \in Var_h \cup Var_{par(h)} \cup Var_{ch(h)}\}$*, being* $par(h)$ *the parent of* h *and* $ch(h)$ *the set of child of* h *in* μ*;*
   - $e(F) \in E_h$ *is an enzyme and* $\alpha(F) \in P$ *is a protein, both of them associated with program* F*, if no enzyme or protein is used in a program then it will be omited;*
9. $\mathcal{R}$ *is a finite set of rules of the following form:*
   - Protein evolution rules*:* $[\, \alpha \to \alpha' \,]_h$*, where* $h \in H, \alpha \in P$ *and* $\alpha' \in P$*.*
   - Writing-only communication rules between the shared memory and the membranes
   $$(h\,,\, X_h\, / Y_{h,mem}\,,\, mem)^W_\alpha$$
   where $X_h \in Var_h, Y_{h,mem} \in Var_{mem}, \alpha \in P$ *in such manner that there is, at most, one rule for each membrane* $h \in \{1, \ldots p \cdot q\}$*. Variables* $Y_{h,mem}, Y_{h',mem}$ *should be different for two membranes* h, h'*.*
   - Reading-only communication rules between the shared memory and the membranes:
   $$(h\,,\, X_h\, / Y_{mem}\,,\, mem)^R_\alpha$$
   where $X_h \in Var_h, Y_{mem} \in Var_{mem}, \alpha \in P$*. Variable* Y_{mem} *is the same for each* $h \in \{1, \ldots p \cdot q\}$*.*
   - Membrane creation rules:
   $$\left[\, \left[\, X_{1,h}\,,\, X_{2,h},\, ,\, \ldots\, ,\, X_{n,h} \,\right]_h \,\right]_{h'}\, ;\, \alpha$$
   where $h, h' \in \{1, \ldots p \cdot q\}$ *are different,* $\alpha \in P$ *and* $X_{1,h}\,,\, \ldots\,,\, X_{n,h} \subseteq Var_h$*.*

The term *region* h ($h \in H$) is used to refer to membrane h in the case $h \in \{1, \ldots, p \cdot q\}$, as well as to refer to the shared memory in the case $h = mem$.

Next, we describe the semantics of RENPSHs. A *configuration* of a RENPSH at any instant t is described by the current membrane structure, together with

proteins and all values of the variables and enzymes associated with all regions (compartments of the current membrane structure and the shared memory). The *initial configuration* is $(\mu, E_{mem}(0), \{P_h(0), Var_h(0) | h \in H\})$, where $\mu = \{h_0\}$.

A program $F(x_{1,h}, \ldots, x_{k_F,h}) \xrightarrow{e(F);\alpha(F)} c_1|v_1, \ldots, c_{n_F}|v_{n_F}$ associated with a region is applicable to a configuration $\mathcal{C}_t$, at moment t, if the value of $e(F)$ at that instant is greater than $\min\{x_{1,h}(t), \ldots, x_{k_F,h}(t)\}$ and protein $\alpha(F)$ is inside the region h of $\mathcal{C}_t$. When applying such a program, variables associated with configuration $\mathcal{C}_t$ are processed as follows: first, the value $F(x_{1,h}(t), \ldots, x_{k_F,h}(t))$ is computed as well as the value

$$q(t) = \frac{F(x_{1,h}(t), \ldots, x_{k_F,h}(t))}{c_1 + \cdots + c_{n_F}}$$

This value represents the *unary portion* at instant t to be distributed among variables $v_1, \ldots, v_{n_h}$ according to the repartition expression. Thus, $q(t) \cdot c_s$ is the contribution added to the current value of v_s $(1 \leq s \leq n_h)$, at step $t+1$. So, $v_s(t+1) = v_s(t) + q(t) \cdot c_s$ and $v_s(t) = 0$, i.e, it is assumed that variable v_s is "consumed" (become zero) when the production function is used and other variables retain their values. Each program in each membrane can only be used once in every computation step, and all the programs are executed in parallel manner.

A protein evolution rule $[\,\alpha \rightarrow \alpha'\,]_h$ is applicable to a configuration $\mathcal{C}_t$ at moment t if protein α is in membrane h of $\mathcal{C}_t$. When applying such a rule the protein α in h evolves to protein α' in h. These rules are applied in a maximal manner.

A writing-only communication rule between the shared memory and the membranes, $(h\,,\, X_h\,/Y_{h,mem}\,,\, mem)^W_\alpha$, is applicable to a configuration $\mathcal{C}_t$ at moment t if protein α is in membrane h of $\mathcal{C}_t$. When applying such a rule the value $X_h(t)$ is assigned to the variable $Y_{h,mem}(t+1)$ of the shared memory, that is $Y_{h,mem}(t+1) \leftarrow X_h(t)$. These rules are applied in a maximal manner.

A reading-only communication rule between the shared memory and the membranes, $(h\,,\, X_h\,/Y_{mem}\,,\, mem)^R_\alpha$ is applicable to a configuration $\mathcal{C}_t$ at moment t if protein α is in membrane h of $\mathcal{C}_t$. When applying such a rule the value $Y_{mem}(t)$ is assigned to the variable $X_h(t+1)$ of membrane h, that is $X_h(t+1) \leftarrow Y_{mem}(t)$. These rules are applied in a maximal manner.

A membrane creation rule $\big[\,\big[\,X_{1,h}\,,\, X_{2,h},\, ,\, \ldots\,,\, X_{n,h}\,\big]_h\,\big]_{h'}$; α is applicable to a configuration $\mathcal{C}_t$ at moment t if protein α is in membrane h' of $\mathcal{C}_t$. When applying such a rule, a new membrane labelled by h is created in such manner that h' is the parent of h and the set of its variables is $Var_h = \{X_{1,h}\,,\, \ldots\,,\, X_{n,h}\}$.

Given a random enzymatic numerical P system with proteins and shared memory Π, we say that configuration $\mathcal{C}_t$ at time t yields configuration $\mathcal{C}_{t+1}$ in one transition step if we can pass from $\mathcal{C}_t$ to $\mathcal{C}_{t+1}$ by applying in parallel each program in each membrane only once, and by applying the rules in a maximal parallel way following the previous remarks. A *computation* of Π is a (finite or infinite) sequence of configurations such that: (a) the first term is the initial configuration of the system; (b) for each $n \geq 2$, the n-th configuration of the

sequence is obtained from the previous configuration in one transition step; and (c) if the sequence is finite (called *halting computation*) then the last term is a *halting configuration* (a configuration where no rule of the system is applicable to it). All the computations start from an initial configuration and proceed as stated above; only halting computations give a result, which is encoded by the objects present in the output region i_{out} associated with the halting configuration. If $\mathcal{C} = \{\mathcal{C}_t\}_{t<r+1}$ of Π ($r \in \mathbb{N}$) is a halting computation, then the *length* of $\mathcal{C}$, denoted by $|\mathcal{C}|$, is r. For each i ($1 \leq i \leq q$), we denote by $\mathcal{C}_t(i)$ the finite multiset of objects over Γ contained in all membranes labelled by i at configuration $\mathcal{C}_t$.

4 Simulation of one iteration of the RRT algorithm

The input of an algorithm generating a Rapidly-Exploring Random Tree $\mathcal{T} = (V_{\mathcal{T}}, E_{\mathcal{T}})$ consists of the following parameters $(x_{init}, K, \Delta t, X, X_{obs}, d_{min})$, where:

- x_{init} is the initial state.
- K is the number of iterations to build the RRT.
- Δt is a fixed amount of time for transitions.
- X is the state space.
- $X_{obs} \subseteq X$ is the obstacle state space.
- d_{min} is the minimum distance threshold according to some distance metric ρ in order to include a new node in the RRT.

For holonomic robots in a 2D environment the state space can be given by the Cartesian coordinates (x, y) of the possible robot positions. For instance, by means of a grid with $p \geq 1$ columns and $q \geq 1$ rows. In this case, any state or position $(i, j) \in \{1, \ldots p\} \times \{1, \ldots q\}$ can be encoded by the natural number $(i-1) \cdot q + j$. In such a manner that, given a natural number n encoding a state (i, j), the following holds: $i = 1 + qt(n, q)$ and $j = rm(n, q)$.

Initially, $V_{\mathcal{T}} = \{x_{init}\}$ and $E_{\mathcal{T}} = \emptyset$ and for each iteration a new node and a new arc can be added to $\mathcal{T}$ according the following scheme:

- A new state x_{rand} is randomly selected by using a function RANDOM_STATE(X).
- The node x_{near} in $\mathcal{T}$ closest to x_{rand} is selected by means of the function NEAREST_NEIGHBOR($x_{rand}, \mathcal{T}, Threshold$) in such manner that if the minimum distance is lesser or equal to $Threshold$, then no node is selected in this iteration.
- The unitary vector U from node x_{near} to node x_{rand} is computed.
- The new node X_{new} is obtained by applying the unitary vector U to the robot for Δt time starting at state x_{near}, that is, $X_{new} = x_{near} + U \cdot \Delta t$. This node will be obtained by using the function SELECT_NEW_NODE($x_{near}, U, \Delta t$).
- The node X_{new} and the arc (X_{near}, X_{new}) is added to the tree $\mathcal{T}$.

This algorithm will be simulated by a random enzymatic numerical P system with proteins and shared memory of degree (p, q)

$$\Pi = (H, \mu, P, E_{mem}, E_{mem}(0), \{(P_h(0), Var_h, Var_h(0), Pr_h) \mid h \in H\}, \mathcal{R}, h_0)$$

defined as follows:

- $H = \{1, \ldots, p \cdot q\} \cup \{mem\}$, $mem \notin \{1, \ldots, p \cdot q\}$.
- $\mu = \{h_0\}$ with $h_0 \in \{1, \ldots, p \cdot q\}$.
- $P = \{\alpha_i \mid 1 \leq i \leq 12\}$, and $P_h(0) = \{\alpha_1\}$, for each $h \in H$.
- $E_{mem} = \{Flag_{mem}, p \cdot q + 1\}$ and $E_{mem}(0) = \{p \cdot q + 1\}$.
- The set of variables is:
  - $Var_h = \{X_{1,h}, X_{2,h}, Y_{1,h}, Y_{2,h}, D_h\}$, for each $h, 1 \leq h \leq p \cdot q$.
  - $Var_{mem} = \{X_{1,mem}, X_{2,mem}, Y_{1,mem}, Y_{2,mem}, Z_{1,mem}, Z_{2,mem}\} \cup$
    $\{U_{1,mem}, U_{2,mem}\} \cup \{D_{h,mem}, Y_{h,mem} \mid 1 \leq h \leq p \cdot q\}$.
  - Initially, all variables in $Var_h (h \neq h_0)$ and all variables in Var_{h_0} different to Y_{1,h_0}, Y_{2,h_0}, are equal to zero. Besides, initially the values (Y_{1,h_0}, Y_{2,h_0}) provide the 2D-coordinates of the "initial state" h_0.

- Next, the finite set of programs P_{r_h} and the set of rules $\mathcal{R}$ of the system are defined according with the requirements to simulate the RRT algorithm.
- In order to synchronize the sequence of an iteration, the following protein evolution rules for each $h \in H$ are considered: $[\, \alpha_i \to \alpha_{i+1} \,]_h$, for $1 \leq i \leq 11$, and $[\, \alpha_{12} \to \alpha_1 \,]_h$
- Two random numbers i, j $(1 \leq i \leq p, 1 \leq j \leq q)$ are generated in the shared memory.

$$\begin{cases} \textit{Production function} : F(X_{1,mem}) = Random(i, 1 \leq i \leq p) \\ \textit{Repartition protocol} : 1|X_{1,mem} \\ \textit{Protein} : \alpha_1 \end{cases}$$

$$\begin{cases} \textit{Production function} : F(X_{2,mem}) = Random(i, 1 \leq i \leq q) \\ \textit{Repartition protocol} : 1|X_{2,mem} \\ \textit{Protein} : \alpha_1 \end{cases}$$

- Each membrane $h \in \{1, \ldots, p \cdot q\}$ will read the random numbers previously generated, sharing them with the variables $X_{1,h}, X_{2,h}$.

$$\begin{cases} (h\,,\, X_{1,h} \,/\, X_{1,mem}\,,\, mem)^R_{\alpha_2} \\ \\ (h\,,\, X_{2,h} \,/\, X_{2,mem}\,,\, mem)^R_{\alpha_2} \end{cases}$$

- For each membrane $h \in \mu$, the distance D_h between its position $(Y_{1,h}, Y_{2,h})$ and the position given by the generated random natural numbers $(X_{1,mem}, X_{2,mem})$ is computed. For the remaining membranes, $D_h = p \cdot q + 1$.

$$\begin{cases} \textit{Production function} : F(X_{1,h}, X_{2,h}, Y_{1,h}, Y_{2,h}) = \begin{cases} \sqrt{\sum_{j=1}^{2}(X_{j,h} - Y_{j,h})^2} & \text{if } h \in \mu \\ p \cdot q + 1 & \text{if } h \notin \mu \end{cases} \\ \\ \textit{Repartition protocol} : 1|D_h \\ \textit{Protein} : \alpha_3 \end{cases}$$

– Each membrane h writes its value D_h to the shared memory.
$(h\,,\, D_h \,/\, D_{h,mem}\,,\, mem)^W_{\alpha_4}$

– The minimum of all distances D_h is computed in the shared memory.
- *Production function*: $F(D_{1,mem}, \ldots, D_{p\cdot q,mem}) = \min\{D_{1,mem}, \ldots, D_{p\cdot q,mem}\}$
- *Repartition protocol*: $1|D_{min,mem}$
- *Protein*: α_5

– Variable (enzyme) $Flag_{mem}$ is set to zero if $D_{min,mem} \leq Threshold$.
- *Production function*: $F(D_{1,mem}, \ldots, D_{p\cdot q,mem}) = \begin{cases} 0 & \text{if } D_{min,mem} \leq Threshold \\ p \cdot q + 1 & \text{otherwise} \end{cases}$
- *Repartition protocol*: $1|Flag_{mem}$
- *Protein*: α_6

– The label *near*, corresponding to the closer membrane to the randomly generated position, is obtained.
- *Production function*: $F(D_{1,mem}, \ldots, D_{p\cdot q,mem}) = \text{arg-min}\{D_{1,mem}, \ldots, D_{p\cdot q,mem}\}$
- *Repartition protocol*: $1|Y_{near,mem}$
- *Protein*: α_7
- *Enzyme*: $Flag_{mem}$

– The position of membrane *near* is computed.

$$\begin{cases} Production\ function : F(Y_{near,mem}) = 1 + qt(Y_{near,mem}, q) \\ Repartition\ protocol : 1|Y_{1,mem} \\ Protein : \alpha_8 \\ Enzyme : Flag_{mem} \end{cases}$$

$$\begin{cases} Production\ function : F(Y_{near,mem}) = rm(Y_{near,mem}, q) \\ Repartition\ protocol : 1|Y_{2,mem} \\ Protein : \alpha_8 \\ Enzyme : Flag_{mem} \end{cases}$$

– The unitary vector is created in the shared memory.

$$\begin{cases} Production\ function : F(X_{1,mem}, X_{2,mem}, Y_{1,mem}, Y_{2,mem}) = \frac{X_{1,mem} - Y_{1,mem}}{\sqrt{\sum_{j=1}^{2}(X_{j,mem} - Y_{j,mem})^2}} \\ Repartition\ protocol : 1|U_{1,mem} \\ Protein : \alpha_9 \\ Enzyme : Flag_{mem} \end{cases}$$

$$\begin{cases} Production\ function : F(X_{1,mem}, X_{2,mem}, Y_{1,mem}, Y_{2,mem}) = \frac{X_{2,mem} - Y_{2,mem}}{\sqrt{\sum_{j=1}^{2}(X_{j,mem} - Y_{j,mem})^2}} \\ Repartition\ protocol : 1|U_{2,mem} \\ Protein : \alpha_9 \\ Enzyme : Flag_{mem} \end{cases}$$

– The position of the new membrane is computed in the shared memory.

$$\begin{cases} \textit{Production function} : F(Y_{1,mem}, U_{1,mem}) = Y_{1,mem} + U_{1,mem} \cdot \Delta t \\ \textit{Repartition protocol} : 1|Z_{1,mem} \\ \textit{Protein} : \alpha_{10} \\ \textit{Enzyme} : Flag_{mem} \end{cases}$$

$$\begin{cases} \textit{Production function} : F(Y_{2,mem}, U_{2,mem}) = Y_{2,mem} + U_{2,mem} \cdot \Delta t \\ \textit{Repartition protocol} : 1|Z_{2,mem} \\ \textit{Protein} : \alpha_{10} \\ \textit{Enzyme} : Flag_{mem} \end{cases}$$

– The membrane labelled by $Y_{near,mem}$ will read the position (Z_1, Z_2) corresponding to the new membrane from the shared memory.

$$\begin{cases} (Y_{near,mem}\ ,\ Z_{1,Y_{near,mem}}\ /\ Z_{1,mem}\ ,\ mem)^R_{\alpha_{11}} \\ \\ (Y_{near,mem}\ ,\ Z_{2,Y_{near,mem}}\ /\ Z_{2,mem}\ ,\ mem)^R_{\alpha_{11}} \end{cases}$$

– A child membrane with position (Z_1, Z_2) is created in Y_{near}, that is, a new state is added to the tree.

$$\left[\left[\begin{matrix} X_{1,h} & X_{2,h} \\ Y_{1,h} & Y_{2,h} \\ Z_{1,h} & Z_{2,h} \\ D_h & \end{matrix} \right]_h \right]_{Y_{near,mem}}$$

Being $h = (Z_{1,Y_{near,mem}} - 1) \cdot q + Z_{2,Y_{near,mem}}$. This rule is mediated by protein α_{12}.

5 Conclusions

This paper deals with an algorithm widely used to solve the problem of motion planning in robots, e.g., the *rapidly-exploring random tree* (RRT) generation algorithm. It is based on the randomized exploration of the configuration space. We have studied it within the framework of Membrane Computing.

In this work, a variant of Enzymatic Numerical P systems, called *random enzymatic numerical P systems with proteins and shared memory* (RENPSM, for short) is introduced. Besides, a simplified version of the standard RRT algorithm is described by a RENPSM system capturing the semantics of the new variant, where maximal parallelism is used.

Three challenges are planned as future work. First, to provide a formal verification of such RENPSM systems, in the sense that they in fact simulate the RRT generation algorithm. The second challenge is to design a software platform in order to simulate RENPSM systems and check the efficiency of such simulation in comparison with the usual implementations of the RRT algorithm. Finally, to provide the simulation of a more real-life RRT algorithm for path planning in robots, such as the case of the bidirectional RRT based planner [6] for non-holonomic robots with kynodinamic and environment constraints.

References

1. K.J. Astrom, T. Hagglund. *PID Controllers: Theory, Design, and Tuning*, 1995
2. M.A. Colomer, A. Margalida, D. Sanuy, and M.J. Pérez-Jiménez. A bio-inspired computing model as a new tool for modeling ecosystems: The avian scavengers as a case study. *Ecological Modelling* 222 (1), 2011, pp. 3347.
3. S. Huang, G. Dissanayake. *Robot Localization: An Introduction.* 2016
4. J.C. Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, Boston, MA, 1991.
5. S.M. LaValle. Rapidly-Exploring Random Trees: A New Tool for Path Planning, *Computer Science Dept.*, Iowa State University, October 1998
6. S.M. LaValle, and J.J. Kuffner. Randomized kinodynamic planning. *Proceedings IEEE International Conference on Robotics and Automation*, 1999, pages 473–479
7. A. Nash, K. Daniel, S. Koenig, and A. Felner. Theta*: Any-Angle Path Planning on Grids. *Journal of Artificial Intelligence Research*, Vol. 39, 2010, pp. 533-579.
8. Gh. Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61 (1), 2000, pp. 108–143.
9. Gh. Păun, R. Păun. Membrane Computing and Economics: Numerical P Systems. *Fundamenta Informaticae*, 73 (1,2). 2006. pp. 213–227
10. A. Pavel, O. Arsene, C. Buiu. Enzymatic Numerical P Systems - A New Class of Membrane Computing Systems *Proceedings of IEEE fifth international conferenced on bio-inspired computing: Theories and applications (BIC-TA)*, 2010, pp. 1331–1336
11. A. Pavel, C. Vasile, I. Dumitrache. Robot localization implemented with enzymatic numerical P systems *Proceedings of the international conference on biomimetic and biohybrid systems*, 2012, pp. 204–215
12. A. Pavel, C. Buiu. Using enzymatic numerical P systems for modeling mobile robot controllers. *Natural Computing*, 11 (3), 2012, pp. 387–393
13. M.J. Pérez-Jiménez. *The P versus NP problem from Membrane Computing view.* European Review, Vol. 22 (1), 2014, pp. 18–33
14. F.J. Romero-Campero, M.J. Pérez-Jiménez. A Model of the Quorum Sensing System in Vibrio fischeri Using P Systems. *Artificial Life*, 14 (1), 2008. 95 – 109
15. A. Stentz. The Focussed D* Algorithm for Real-time Replanning. *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, Vol. 2, 1995, pp. 1652–1659
16. H. Wang, Y. Yu, and Q. Yuan. Application of Dijkstra algorithm in robot path-planning, *Proceedings of the 2nd International Conference on Mechanic Automation and Control Engineering.* 2011, pp. 1067–1069.
17. T. Wang, G. Zhang, J. Zhao, Z. He, J. Wang, and M.J. Pérez-Jiménez. Fault diagnosis of electric power systems based on fuzzy reasoning spiking neural P systems. *IEEE Transactions on Power Systems* 30 (3), 2015. pp. 11821194.
18. G. Zhang, M.J. Perez-Jimenez, M. Gheorghe. *Real-life Applications with Membrane Computing.* 2016