Membrane Computing Meets Artificial Intelligence: A Case Study

Miguel A. Gutiérrez-Naranjo, Mario J. Pérez-Jiménez

Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012, Sevilla, Spain
magutier@us.es, marper@us.es

Summary. The usual way to find a solution for a NP complete problem with Membrane Computing techniques is by brute force algorithms where all the feasible solutions are generated and they are checked simultaneously by using massive parallelism. These solutions work from a theoretical point of view but they are implementable only for small instances of the problem. In this paper we provide a family of P systems which brings techniques from Artificial Intelligence into Membrane Computing and apply them to solve the N-queens problem.

1 Introduction

Brute force algorithms have been widely used in the design of solutions for NP problems in Membrane Computing. Trading time against space allows us to solve NP problems in polynomial (even lineal) time with respect to the input data. The cost is the number of resources, mainly the number of membranes, which grows exponentially. The usual idea of such brute force algorithms is to encode each feasible solution in one membrane. The number of candidates to solution is exponential in the input size, but the coding process can be done in polynomial time. Once generated all these candidates, each of them is tested in order to check whether it represents a solution to the problem or not. This checking stage is made simultaneously in all membranes by using the massive parallelism inherent to Membrane Computing. Any computational device that performs this checking sequentially needs an exponential amount of time. After the checking stage ends, the P system halts and sends a signal to the user with the output of the process.

Such theoretical process works and many different P system models have been explored by searching the limits between tractability and intractability [2]. In such way, several semantic and syntactic ingredients have been mixed and nowadays there exist many open questions and open problems in the area (see, e.g., [6]).

In spite of the great success in the design of theoretical solutions to NP problems, these solution have an intrinsic drawback from a practical point of view. It is not clear yet whether Membrane Computing will have an *in vitro*, *in vivo* or *in silico* implementation, but in any case, a membrane will have a *space* associated (maybe a piece of memory in a computer, a pipe in a lab or the volume of a bacteria) and brute force algorithms only will be able to implement little instances of such problems. As an illustration, if we consider an in vivo implementation where each feasible solution is encoded in an elementary membrane and such elementary membrane is *implemented* in a bacteria of mass similar to E. Coli ($\sim 7 \times 10^{-16}$ kg., see [8]), then, a brute force algorithm[1] which solve an instance of a NP problem with input size 40 will need approximately the mass of the Earth for an implementation ($\sim 6 \times 10^{24}$ kg., *ibid.*).

In this paper we explore the possibility of searching solutions to NP problems with Membrane Computing techniques, but taking ideas from Artificial Intelligence instead of using brute force algorithms. Of course, the worst case of any solution of an NP-problem needs and exponential amount of resources, but we are not always in the worst case. The contribution of using search strategies from Artificial Intelligence is that, on average, the number of resources for solving several instances of an NP problem decreases with respect to the number of resources used by brute force, since an exponential number of resources is always used in the former one. The case study is the N-queens problem (Section 2), which was previously studied in the framework of Membrane Computing in [3].

The paper is organized as follows: Next we present the N-queens problem and recall the brute force algorithm presented in [3]. In Section 3, we give a brief notions of searching strategies in Artificial Intelligence and in Section 4, we present an implementation of depth-first search with P systems. In Section 5, we present a family of P systems which solve the N-queens problem based on the cellular implementation. Finally, some conclusions and new open research lines are presented.

2 The N-queens Problem

Along this paper we will consider the N-queens problem as a case study. The N-queens problem is very popular among computer scientists. It is a generalization of a classic problem known as the 8-queens problem. The original one is attributed to the chess player Max Bezzel and it consists on putting eight queens on an 8×8 chessboard in such way that none of them is able to capture any other using the standard movement of the queens in chess, i.e., at most one queen can be placed on each row, column and diagonal line.

The 8-queens problem was later generalized to the N-queens problem, with the same rules but placing N queens on a N×N board.

[1] A similar comparison was proposed by Niall Murphy during the Tenth Workshop on Membrane Computing.

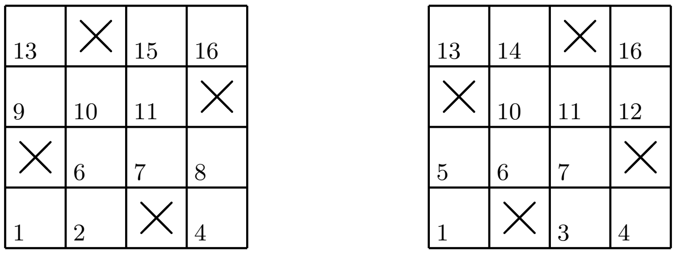


Fig. 1. Solutions to the 4-queens problem

In [3], a first solution to the N-queens problem in Membrane Computing was shown. For that aim, a family of deterministic P systems with active membranes was presented. In such family, the N-th element of the family solves the N-queens problem and the last configuration encodes *all* the solutions of the problem.

In order to solve the N-queens problem, a truth assignment such that it makes true a formula in CNF is searched. This problem is exactly SAT, so the solution presented in [3] uses a modified solution for SAT from [5].

In such a paper, it was proven that given an integer $N \geq 3$, there exists a formula Φ in conjunctive normal form such that encodes the N-queens problem with N^2 variables and $\frac{1}{3}(5N^3 - 6N^2 + 4N)$ clauses.

Some experiments were presented by running the corresponding P systems with an updated version of the the P-lingua simulator [1]. The experiments were performed on a one-processor Intel core2 Quad (with 4 cores at 2,83Ghz), 8GB of RAM and using a C++ simulator over the operating system Ubuntu Server 8.04.

In the 3-queens problem, three queens should be placed on a 3×3 chessboard. According to our representation, the problem can be expressed by a formula in CNF with 9 variables and 31 clauses. The *input multiset* has 65 elements and the P system has 3185 rules. Along the computation, $2^9 = 512$ elementary membranes need to be considered in parallel. Since the simulation was carried out in a one-processor computer, in the simulation, these membranes were evaluated sequentially. It took 7 seconds to reach the halting configuration. It is the 117-th configuration and in this configuration one object `No` appears in the environment. As expected, this means that we cannot place three queens on a 3×3 chessboard satisfying the restriction of problem.

In the 4-queens problem, we try to place four queens on a 4×4 chessboard. According to our representation, the problem can be expressed by a formula in CNF with 16 variables and 80 clauses. Along the computation, $2^{16} = 65536$ elementary membranes were considered in the same configuration and the P system has 13622 rules.

The simulation takes 20583 seconds (> 5 hours) to reach the halting configuration. It is the 256-th configuration and in this configuration one object `Yes` appears in the environment. This means that there exists at least one solution to the problem. In order to know such solutions, we check the multiset of the elementary membranes. In this case there are two elementary membranes in the halting configuration with the following multisets:

$$w_1 = \{f_1, f_2, t_3, f_4, t_5, f_6, f_7, f_8, f_9, f_{10}, f_{11}, t_{12}, f_{13}, t_{14}, f_{15}, f_{16}\}$$

$$w_2 = \{f_1, t_2, f_3, f_4, f_5, f_6, f_7, t_8, t_9, f_{10}, f_{11}, f_{12}, f_{13}, f_{14}, t_{15}, f_{16}\}$$

Such multisets encode the solution showed in the Figure 1

According to this design, for the solution of the N-queens problem in a standard 8×8 chessboard 2^{64} = 18.446.744.073.709.551.616 elementary membranes should be considered simultaneously. If we follow with the analogy from the Introduction, an *E. Coli implementation* of such P system we will need approximately a metric tone of bacteria to solve the problem. Does it means that Membrane Computing is not able to find at least one solution to the N-queens problem on a 8×8 chessboard?

3 Searching Strategies

Searching has been deeply studied in Artificial Intelligence. In its basic form, a *state* is an instantaneous description of the world and two states are linked by a *transition* which allows us to reach a state from a previous one. In such way, we consider a directed graph where the nodes are the states and the edges are the actions. Giving a starting state, a sequence of actions (a path in a graph) to one of the final states is searched.

In sequential algorithms, only one node is considered in each time unit and the order in which we explore new nodes determines the different searching strategies. In the usual framework, several possible unexplored nodes are reachable and we need to choose one of them in order to continue the search. In the best case, we have a heuristic which can help us to decide the best options among the candidates. Such heuristic represents, in a certain sense, how far the considered node is from a solution node and it captures our information about the nature of the problem. In many other situations we have no information about how far we are from a solution and we need to use a *blind strategy.*

Since there is no information about the nature of the problem, blind strategies are based exclusively in the topology of the graph and the order in which new nodes are reached.

There exists a clear parallelism between the space of states represented as a directed graph and the computation trees in Membrane Computing. In Membrane Computing, we start with an initial description of the world (the initial configuration) and, in the general case, we have several sets of applicable rules which lead us to different configurations. We choose one of the reachable configurations and go on with the process till reaching a halting configuration. In the case of recognizer P systems, no matter which new configuration we choose among the different possibilities since all of them lead us to the same answer, but this is not the general case.

The two basic blind search strategies are depth-first search and breath-first search. The main difference between them is that depth-first search follows a path to its completion before trying an alternative path. Some path can be infinite, so this search may never succeed. It involves *backtracing*: One alternative is selected for each node and it backtracks to the next alternative when it has pursued all of the paths from the first choice. In the worst case, depth-first search will explore all of the $O(b^m)$ nodes in the search tree, where m is the maximum depth of any node and b is the maximum branching factor. The complexity in time is $O(b^m)$, and the complexity in space is $O(bm)$.

In breath-first search the order in which nodes are explored depends on the number of arcs in the path. The algorithm always selects one of the paths with fewest arcs. If there exists a solution at depth d, the total number of generated nodes is $O(b^{d+1})$. In this case the complexity in time and in space is $O(b^{d+1})$.

4 Depth-first Search with P Systems

The idea of representing an instantaneous description of the world as a state and a step from a state to the following one as an edge in the graph is so general that many real-life problems can be modeled as a problem of space of states. In this paper we present a first approach to depth-first search with P systems. The aim is to show that Membrane Computing provides all the ingredients that we need to find a solution for any problem represented as a space of states and hence, to be a useful tool to solve many real-life problems.

The aim of this first approach is not minimalist. We are not looking for the minimum number of ingredients for implementing in P systems the depth-first search. In fact, we use four of the most powerful available ingredients: inhibitors, cooperation, priorities and dissolution. As we will remark in Section 6, it is an open question to weaken these conditions.

In an abstract way, the representation of a problem $P = (a, S, E, F)$ as a space of states consists on:

- A set of states S and an initial state, $a \in S$
- A set E of ordered pairs (x, y), called *transitions*, where x and y are states and y is reachable from x in one step.
- A set F of final states.

Technically, we also need a *cost* mapping, which assigns a cost to each transition (x, y), but we will consider a constant cost and we will omit it.

Given a problem $P = (a, S, E, F)$, we will consider a P system $\Pi = (\Gamma, H, \mu, w_e, w_s, R_1, R_2, R_3, R_1 > R_2, > R_3)$ where

- The alphabet $\Gamma = S \cup \{p_e, r_e \mid e \in E\}$
- The set of labels $H = \{u, s\}$
- A membrane structure $\mu = [\,[\,]_u\,]_s$
- The initial multisets $w_u = \{a\}$ and $w_s = \emptyset$.

- Three sets of rules R_1, R_2 and R_3
  - $R_1 = \{[x]_u \rightarrow \lambda : x \in F\}$. For each final state we have a dissolution rule which dissolves the membrane u.
  - $R_2 = \{[x \neg p_y \rightarrow y\, r_{xy}]_u \, : (x, y) \in E\}$. For each transition (x, y), x can be changed by $y\, r_{xy}$ if p_y does not occur in the membrane u, i.e., p_y acts as an inhibitor.
  - $R_3 = \{[y\, r_{xy} \rightarrow x\, p_y]_u \, : \, (x, y) \in E\}$. For each transition (x, y) we have a co-operative rule where the multiset $y\, r_{xy}$ is rewritten as $x\, p_y$ in the membrane u.
- Finally, we have an order among the rules. Rules of R_1 have priority over the other rules and rules from R_2 have priority over rules from R_3.

The intuition behind the objects is the following: In each configuration (but in the last one) there is one object from S in the configuration. It represents the current state in the searching process. For each state y, the object p_y is an inhibitor[2] which forbid to visit the state y. Finally, the occurrence of the object r_{xy} represents that the transition (x, y) belongs to the path from the initial state to the current one. We illustrate one computation of these P systems with the following example.

4.1 Example

Let us consider the space of states $P = (a, S, E, F)$ with a the initial state, the set of transitions $E = \{(a, b), (a, c), (b, d), (b, e), (e, f), (c, g)\}$ and the set of final states $F = \{g\}$. Let Π be the P system associated with this space as described above. The initial configuration is $C_0 = [\,[a]_u\,]_s$. Two rules are applicable, both belonging to the set R_2, $r_b \equiv [a \neg p_b \rightarrow b\, r_{ab}]_u$ and $r_c \equiv [a \neg p_c \rightarrow c\, r_{ac}]_u$. Let us suppose that non-deterministically r_b is chosen. Then we have $C_1 = [\,[b\, r_{ab}]_u\,]_s$. From C_1, three rules are applicable

$$r_d \equiv [b \neg p_d \rightarrow d\, r_{bd}]_u \in R_2 \quad r_e \equiv [b \neg p_e \rightarrow e\, r_{be}]_u \in R_2 \quad r_{\underline{b}} \equiv [b\, r_{ab} \rightarrow a\, p_b]_u \in R_3$$

Since R_2 has priority over R_3, only r_d or r_e can be non-deterministically chosen. We choose r_e and reach $C_2 = [\,[e\, r_{ab}\, r_{be}]_u\,]_s$. Now, only two rules are applicable

$$r_f \equiv [e \neg p_f \rightarrow f\, r_{ef}]_u \in R_2 \qquad r_{\underline{e}} \equiv [e\, r_{be} \rightarrow b\, p_e]_u \in R_3$$

Since R_2 has priority, r_f is applied and we reach $C_3 \equiv [\,[f\, r_{ab}\, r_{be}\, r_{ef}]_u\,]_s$. From C_3, the unique applicable rule is $r_{\underline{f}} \equiv [f\, r_{ef} \rightarrow e\, p_f]_u \in R_3$ and $C_4 \equiv [\,[e\, r_{ab}\, r_{be}\, p_f]_u\,]_s$. Notice than the application of $r_{\underline{f}}$ is an implementation of backtracing. In the configuration C_4, the current state is e and the state f is forbidden. From C_4, only $r_{\underline{e}} \equiv [e\, r_{be} \rightarrow b\, p_e]_u \in R_3$ is applicable. The application of this rule is a new step of backtracing and it leads us to the configuration $C_5 \equiv [\,[b\, r_{ab}\, p_e\, p_f]_u\,]_s$. From C_5, two rules are applicable

[2] Notice taht the object p_y is never removed. If the state y can be reached from different paths, then we should add new rules in order to prevent it.

$$r_d \equiv [b\,\neg p_d \rightarrow d\,r_{bd}]_u \in R_2 \qquad r_{\underline{b}} \equiv [b\,r_{ab} \rightarrow a\,p_e]_u \in R_3$$

Notice that the rule $r_e \equiv [b\,\neg p_e \rightarrow e\,r_{be}]_u \in R_2$ is not applicable due to the occurrence of the inhibitor p_e in the membrane u. Since R_2 has priority over R_3, the rule r_d is applied an the configuration $C_6 \equiv [\,[d\,r_{ab}\,r_{bd}\,p_e\,p_f]_u\,]_s$ is reached. From C_6 we only can do backtracing by applying the rule $r_{\underline{d}} \equiv [d\,r_{bd} \rightarrow b\,p_d]_u \in R_3$ and reach $C_6 \equiv [\,[b\,r_{ab}\,p_d\,p_e\,p_f]_u\,]_s$. By applying now $r_{\underline{b}} \equiv [b\,r_{ab} \rightarrow a\,p_b]_u \in R_3$ we obtain $C_7 \equiv [\,[a\,p_b\,p_d\,p_e\,p_f]_u\,]_s$. From C_7 we only can apply $r_c \equiv [a\,\neg p_c \rightarrow c\,r_{ac}]_u \in R_2$ and reach $C_8 \equiv [\,[c\,r_{ac}\,p_b\,p_d\,p_e\,p_f]_u\,]_s$. From C_8 two rules are applicable

$$r_g \equiv [c\,\neg p_g \rightarrow g\,r_{cg}]_u \in R_2 \qquad r_{\underline{c}} \equiv [c\,r_{ac} \rightarrow a\,p_c]_u \in R_3$$

Due to the priority of R_2 over R_3, r_g is applied and we obtain $C_9 \equiv [\,[g\,r_{ac}\,r_{cg}\,p_b\,p_d\,p_e\,p_f]_u\,]_s$. Finally, the applicable rules are

$$r_F \equiv [g]_u \rightarrow \lambda \in R_1 \qquad r_{\underline{g}} \equiv [g\,r_{cg} \rightarrow c\,p_g]_u \in R_3$$

Since R_1 has priority over R_3, the rule r_F is applied and the configuration $C_{10} \equiv [r_{ac}\,r_{cg}\,p_b\,p_d\,p_e\,p_f]_s$. No more rules are applicable and C_{10} is a halting configuration. The objects r_{ac} and r_{cg} determine a path from the initial state to the final one. Notice that the chosen rules in the non-deterministic points are crucial. From C_0 the configuration $C_3^* \equiv [r_{ac}\,r_{cg}]_s$ is reachable in three steps by applying sequentially the rules $r_c \equiv [a\,\neg p_c \rightarrow c\,r_{ac}]_u \in R_2$, $r_g \equiv [c\,\neg p_g \rightarrow g\,r_{cg}]_u \in R_2$ and $r_F \equiv [g]_u \rightarrow \lambda \in R_1$.

5 A New Solution for the N-queens Problem

The first step for designing a new solution for the N-queens problem is to determine the space of states. There are two basic formulations (see [7]). A *complete-state formulation*, which starts with N queens on the board and moves them around and an *incremental formulation*, where the operators augment the state description, starting from the empty state and each action adds a queen to the state. This second formulation reduces drastically the space of states, since a new queen added to the description of a state can be placed only in a non forbidden square. In such way, states and transitions are the following:

- **States:** Arrangements of k queens ($0 \leq k \leq N$), one per column in the leftmost k columns.
- **Transitions** (x, y)**:** The state y is the state x where a new queen is added in the leftmost empty column. Such new queen is not attacked by any other one.

The basic idea of the P system design is to encode the position of a queen as a set of four objects x_i, y_j, u_{i-j} and v_{i+j}, where x_i represents a column and y_j represents a row ($1 \leq i, j \leq N$). The objects u_{i-j} and v_{i+j} represent the ascendant and the descendant diagonals respectively and their subindices are determined by the corresponding column and row i and j. Placing a queen on the chessboard means to choose a square, i.e., a set $\{x_i, y_j, u_{i-j}, v_{i+j}\}$ among the eligible objects and delete them from the corresponding membrane. The choice is recorded. If the final state is reached we finish the process; otherwise we do backtracing and choose other eligible set.

We present a family of P systems which find a solution for the N-queens problem (a P system for each value of N) slightly different from the general one presented in Section 4. We add a new set of rules R^* for cleaning purposes. For each positive integer greater than 2, we consider the P system

$\Pi = (\Gamma, H, \mu, w_e, w_s, R_1, R^*, R_2, R_3, R_1 > R^* > R_2, > R_3)$ where

- The alphabet $\Gamma = \{x_i, y_j, u_{i-j}, v_{i+j}, p_{i,j} \,:\, i, j \in \{1, \ldots, N\}\} \cup \{x_{N+1}\}$
- The set of labels $H = \{u, s\}$
- The initial multisets $w_u = \{x_1, y_1, \ldots, y_N, u_{1-N}, \ldots, u_{N-1}, v_2, \ldots, v_{2N}\}$ and $w_s = \emptyset$.
- A membrane structure $\mu = [\,[\,]_u\,]_s$
- Four sets of rules R_1, R^*, R_2 and R_3
  - $R_1 = \{[x_{N+1}]_u \to \lambda : x \in F\}$. In this design, when the object k_N is reached, the membrane u is dissolved and the computation ends.
  - $R^* = \{[p_{i,j} x_{i-1} \to x_{i-1}]_u \,:\, i \in \{2, \ldots, N\},\ j \in \{1, \ldots, N\}\}$ Just cleaning rules.
  - $R_2 = \{[x_i\, y_j\, u_{i-j}\, v_{i+j}\, \neg p_{i,j} \to x_{i+1}\, r_{i,j}\,]_u \,:\, i, j \in \{1, \ldots, N\}\}$ These rules put a new queen on the chessboard by choosing an eligible position.
  - $R_3 = \{[r_{i,j}\, x_{i+1} \to x_i\, y_j\, u_{i-j}\, v_{i+j}\, p_{i,j}]_u\ i, j \in \{1, \ldots, N\}\}$. These rules remove one queen form the chessboard and implement the backtracing.
- Finally, the order $R_1 > R^* > R_2, > R_3$ among the sets of rules is settled.

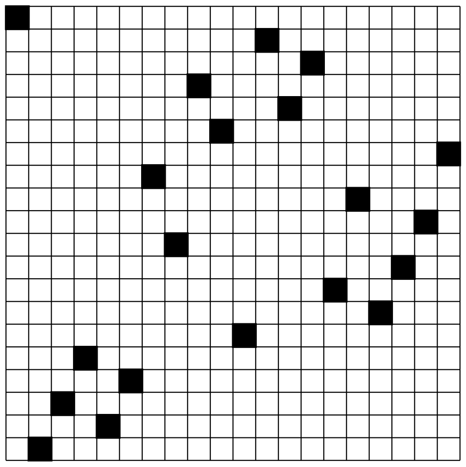
5.1 Hints on the computation

From the objects $\{x_1, \ldots, x_N\}$, only x_1 occurs in the initial configuration. This means that the column 1 is already chosen. In order to take the row, one of the N rules $[k_0\, x_1\, y_j\, u_{1-j}\, v_{1+j}\, \neg p_{1,j,0} \to x_2\, r_{1,j,1}\, k_2]_u$ where $j \in \{1, \ldots, N\}$ is chosen. The election of this rule determines the square (x_1, y_j) where the first queen is placed. The application of the rule removes the objects corresponding to the column, row ascendant and descendant diagonal lines $x_1\, y_j\, u_{1-j}\, v_{1+j}$ in the chessboard. The associated column, row and diagonals to these objects are no eligible and the new queen will be put in a safe square. The application of the rule produces the object x_2. Next, a rule from the set $[k_1\, x_2\, y_j\, u_{2-j}\, v_{2+j}\, \neg p_{2,j,1} \to x_3\, r_{2,j,2}\, k_3]_u$ is chosen. If the successive choices are right, then the object k_N is reached and the membrane u dissolved. The objects $r_{i,j,r}$ in the membrane s from the halting configuration give us a solution to the problem. If no rules from the set R_2 can be applied, then we apply one rule from R_3. As shown in the general case, such rules implement backtracing and produces objects $p_{i,j,r}$ which act as inhibitors. Before applying rules from R_2 or R_3, the P system tries to apply rules from R_1, which means the halt of the computation, or from R^*, which clean useless inhibitor objects.

5.2 Examples

Figure 3 shows a computation of the P system which solves the four queens problem, where the rules are non deterministically chosen. The subindices of the objects $r_{1,2}r_{2,4}r_{3,1}r_{4,3}$ in the halting configuration give us the found solution. In this case the squares for the four queens are $(1,2), (2,4), (3,1), (4,3)$.

An *ad hoc* CLIPS program has been written based on this design of solution for the N-queens problem based on Membrane Computing techniques. Figure 2 shows a solution for the 20-queens problem found by such computer program.



1-20 2-1 3-3 4-5 5-2 6-4 7-13 8-10 9-17 10- 15
11-6 12-19 13-16 14-18 15-8 16-12 17-7 18-9 19-11 20-14

Fig. 2. A solution for the 20-queens problem

6 Conclusions and Future Work

The purpose of this paper is twofold. On the one hand, to stress the inviability of solutions based on brute force algorithms for intractable problems, even in case of a future implementations. On the other hand, to open a door in Membrane Computing to Artificial Intelligence techniques, which are broadly studied and which can enrich the methodology of the design of P system solutions.

This first approach can be improved in many senses. As pointed out in Section 4, the aim of this paper is not minimalist and probably, searching algorithms can be implemented into P systems by using more simple P system models. The second improvement is associated to the nature of P systems. The design of P systems

Applied rule	*Configuration*
	$C_0 \equiv \left[\left[x_1 y_2 y_3, y_4 u_{-3} \ldots u_3 v_2 \ldots, v_8 \right]_u \right]_s$
$[x_1 y_1 u_0 v_2 \neg p_{1,1} \rightarrow x_2 r_{1,1}]_u$	$C_1 \equiv \left[\left[\begin{array}{l} x_2 y_2 y_3 y_4 u_{-3}, u_{-2} u_{-1} \\ u_1 u_2 u_3 v_3 \ldots v_8 r_{1,1} \end{array} \right]_u \right]_s$
$[x_2 y_3 u_{-1} v_5 \neg p_{2,3} \rightarrow x_3 r_{2,3}]_u$	$C_2 \equiv \left[\left[\begin{array}{l} x_3 y_2 y_4 u_{-3}, u_{-2} u_1 u_2 u_3 \\ v_3 v_4 v_6 v_7 v_8 r_{1,1} r_{2,3} \end{array} \right]_u \right]_s$
$[r_{2,3} x_3 \rightarrow x_2\, y_3\, u_{-1}\, v_5\, p_{2,3}]_u$	$C_3 \equiv \left[\left[\begin{array}{l} x_2 y_2 y_3 y_4 u_{-3}, u_{-2} u_{-1} u_1 u_2 \\ u_3 v_3 v_4 v_5 v_6 v_7 v_8 r_{1,1} p_{2,3} \end{array} \right]_u \right]_s$
$[x_2\, y_4\, u_{-2}\, v_6 \neg p_{2,4} \rightarrow x_3\, r_{2,4}]_u$	$C_4 \equiv \left[\left[\begin{array}{l} x_3 y_2 y_3 u_{-3} u_{-1} u_1 u_2 u_3 \\ v_3 v_4 v_5 v_7 v_8 r_{1,1} r_{2,4} p_{2,3} \end{array} \right]_u \right]_s$
$[x_3 y_2 u_1 v_5 \neg p_{3,2} \rightarrow x_4\, r_{3,2}]_u$	$C_5 \equiv \left[\left[\begin{array}{l} x_4 y_3 u_{-3}, u_{-1} u_2 u_3 v_3 v_4 \\ v_7 v_8 r_{1,1} r_{2,4} r_{3,2} p_{2,3} \end{array} \right]_u \right]_s$
$[r_{3,2} x_4 \rightarrow x_3 y_2 u_1 v_5 p_{3,2}]_u$	$C_6 \equiv \left[\left[\begin{array}{l} x_3 y_2 y_3 u_{-3}, u_{-1} u_1 u_2 u_3 v_3 v_4 \\ v_5 v_7 v_8 r_{1,1} r_{2,4} p_{3,2} p_{2,3} \end{array} \right]_u \right]_s$
$[r_{2,4} x_3 \rightarrow x_2 y_4 u_{-2} v_6 p_{2,4}]_u$	$C_7 \equiv \left[\left[\begin{array}{l} x_2 y_2 y_3 y_4 u_{-3} u_{-2} u_{-1} \\ u_1 u_2 u_3 v_3 v_4 v_5 v_6 v_7 v_8 \\ r_{1,1} p_{3,2} p_{2,3} p_{2,4} \end{array} \right]_u \right]_s$
$[p_{3,2} x_2 \rightarrow x_2]_u$	$C_8 \equiv \left[\left[\begin{array}{l} x_2 y_2 y_3 y_4 u_{-3} u_{-2} u_{-1} \\ u_1 u_2 u_3 v_3 v_4 v_5 v_6 v_7 v_8 \\ r_{1,1} p_{2,3} p_{2,4} \end{array} \right]_u \right]_s$
$[r_{1,1} x_2 \rightarrow x_1 y_1 u_0 v_2 p_{1,1}]_u$	$C_9 \equiv \left[\left[\begin{array}{l} x_1 y_1 y_2 y_3 y_4 u_{-3} u_{-2} \\ u_{-1} u_0 u_1 u_2 u_3 v_2 v_3 v_4 v_5 \\ v_6 v_7 v_8 p_{1,1} p_{2,3} p_{2,4} \end{array} \right]_u \right]_s$
$[p_{2,3} x_1 \rightarrow x_1]_u$	$C_{10} \equiv \left[\left[\begin{array}{l} x_1 y_1 y_2 y_3 y_4 u_{-3} u_{-2} \\ u_{-1} u_0 u_1 u_2 u_3 v_2 v_3 v_4 v_5 \\ v_6 v_7 v_8 p_{1,1} p_{2,4} \end{array} \right]_u \right]_s$
$[p_{2,4} x_1 \rightarrow x_1]_u$	$C_{11} \equiv \left[\left[\begin{array}{l} x_1 y_1 y_2 y_3 y_4 u_{-3} u_{-2} \\ u_{-1} u_0 u_1 u_2 u_3 v_2 v_3 v_4 v_5 \\ v_6 v_7 v_8 p_{1,1} \end{array} \right]_u \right]_s$
$[x_1 y_2 u_{-1} v_3 \neg p_{1,2} \rightarrow x_2 r_{1,2}]_u$	$C_{12} \equiv \left[\left[\begin{array}{l} x_2 y_1 y_3 y_4 u_{-3} u_{-2} u_0 u_1 u_2 \\ u_3 v_2 v_4 v_5 v_6 v_7 v_8 p_{1,1} r_{1,2} \end{array} \right]_u \right]_s$
$[x_2 y_4 u_{-2} v_6 \neg p_{2,4} \rightarrow x_3 r_{2,4}]_u$	$C_{13} \equiv \left[\left[\begin{array}{l} x_3 y_1 y_3 u_{-3} u_0 u_1 u_2 u_3 \\ v_2 v_4 v_5 v_7 v_8 p_{1,1} r_{1,2} r_{2,4} \end{array} \right]_u \right]_s$
$[x_3 y_1 u_2 v_4 \neg p_{3,1} \rightarrow x_4 r_{3,1}]_u$	$C_{14} \equiv \left[\left[\begin{array}{l} x_4 y_3 u_{-3} u_0 u_1 u_3 v_2 v_5 v_7 v_8 \\ p_{1,1,0} r_{1,2} r_{2,4} r_{3,1} \end{array} \right]_u \right]_s$
$[x_4 y_3 u_1 v_7 \neg p_{4,3} \rightarrow x_5 r_{4,3}]_u$	$C_{15} \equiv \left[\left[\begin{array}{l} x_5 u_{-3} u_0 u_3 v_2 v_5 v_8 \\ p_{1,1} r_{1,2} r_{2,4} r_{3,1} r_{4,3} \end{array} \right]_u \right]_s$
$[x_5] \rightarrow \lambda$	$C_{16} \equiv \left[\begin{array}{l} u_{-3} u_0 u_3 v_2 v_5 v_8 \\ p_{1,1} r_{1,2} r_{2,4} r_{3,1} r_{4,3} \end{array} \right]_s$

Fig. 3. Example of computation

which computes searching is too close to the classical sequential algorithm. In fact, although the presented P system family uses non-determinism in the choice of the rules, it does not explore the intrinsic parallelism of P systems. The next step in this way is to design algorithms which uses a limited form of parallelism where several rules can be applied simultaneously, but controlling the exponential explosion of brute force algorithms. The current hardware based on Compute Unified Device Architecture [4] from Nvidia can be a clue for these new generation of algorithms.

Acknowledgements

The authors acknowledge the support of the projects TIN2008-04487-E and TIN-2009-13192 of the Ministerio de Ciencia e Innovación of Spain and the support of the Project of Excellence with *Investigador de Reconocida Valía* of the Junta de Andalucía, grant P08-TIC-04200.

References

1. D. Díaz-Pernil, I. Pérez-Hurtado, M.J. Pérez-Jiménez, A. Riscos–Núñez. A P-Lingua Programming Environment for Membrane Computing. *Lecutre Notes in Computer Science*, **5391**, (2009), 187-203.
2. M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos–Núñez, F.J. Romero-Campero. Computational efficiency of dissolution rules in membrane systems. *International Journal of Computer Mathematics* **83**(7), (2006) 593 - 611.
3. M.A. Gutiérrez-Naranjo, M.A. Martínez-del-Amor, I. Pérez-Hurtado, M.J. Pérez-Jiménez. Solving the N-Queens Puzzle with P Systems. Seventh Brainstorming Week on Membrane Computing. Vol I. R. Gutiérrez-Escudero, M.A. Gutiérrez-Naranjo, Gh. Păun, I. Pérez-Hurtado, A. Riscos–Núñez (Eds.). Fénix Editora, Sevilla (2009) 199-210.
4. M.A. Martínez-del-Amor, I. Pérez-Hurtado, Mario J. Pérez-Jiménez, J.M. Cecilia, G. Guerrero, J.M. García. Simulation of Recognizer P Systems by Using Many-core GPUs. Seventh Brainstorming Week on Membrane Computing. Vol II. M.A. Martínez-del-Amor, E.F. Orejuela-Pinedo, Gh. Păun, I. Pérez-Hurtado, A. Riscos–Núñz (Eds.). Fénix Editora, Sevilla (2009) 45-57.
5. M.J. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrinini. A polynomial complexity class in P systems using membrane division. In E. Csuhaj-Varjú, C. Kintala, D. Wotschke, G. Vaszil (Eds.). Proceedings of the 5th Workshop on Descriptional Complexity of Formal Systems, DCFS 2003, Computer and Automaton Research Institute of the Hungarian Academy of Sciences (2003), pp.:284-294.
6. M.J. Pérez-Jiménez. A Computational Complexity Theory in Membrane Computing. *Lecture Notes in Computer Science*, **5957** (2010), 125-148.
7. S. Russell, P. Norvig. Artificial Intelligence. A Modern Approach. Second Edition. Pearson Education, Inc. 2003.
8. Wikipedia. `http://en.wikipedia.org/wiki/Orders_of_magnitude_(mass)`