Depth-First Search with P Systems

Miguel A. Gutiérrez-Naranjo and 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,marper}@us.es

Abstract. The usual way to find a solution for an NP complete problem in Membrane Computing is by brute force algorithms. 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 time with respect to the input data. The cost is the amount of resources, which grows exponentially. The usual idea of these 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 massive parallelism. Next, the P system halts and sends a signal to the user with the output of the process. Such theoretical process works and different P system models have been explored by searching the limits between tractability and intractability [3]. In such way, several ingredients have been mixed and nowadays there exist many 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 solutions have an intrinsic drawback from a practical point of view. In all imaginable implementation, 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 bacterium) 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 bacterium of mass similar to E. Coli ($\sim 7 \times 10^{-16}$ kg., see [9]), then, a brute force algorithm which solves an instance of an 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 an 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. As a case study, we present the N-queens problem (Section 2), previously studied in the framework of Membrane Computing in [2].

The paper is organized as follows: Next we present the N-queens problem and recall the algorithm presented in [2]. In Section 3, we give some brief notions of searching strategies in Artificial Intelligence and in Section 4, an implementation of depth-first search with P systems is shown. 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 open research lines are presented.

2 The N-Queens Problem

Along this paper we will consider the N-queens problem as a case study. It is a generalization of a classic problem known as the 8-queens problem. It consists on putting N queens on an N×N 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.

In [2], 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 this 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 that satisfies a formula in conjunctive normal form (CNF) is searched. This problem is exactly SAT, so the solution presented in [2] uses a modified solution for SAT from [7]. Some experiments were presented by running the P systems with an updated version of the P-lingua simulator [1]. The experiments were performed on a system with an Intel Core2 Quad CPU (a single processor with 4 cores at 2,83Ghz), 8GB of RAM and using a C++ simulator under the operating system Ubuntu Server 8.04.

According to the representation in [2], the 3-queens problem is 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 on a uniprocessor system, 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 the problem.

In the 4-queens problem, we try to place four queens on a 4×4 chessboard. According to the 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 configuration has two elementary membranes encoding the two solutions of the problem (see [2] for details).

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 will need approximately a metric ton of bacteria to solve the problem.

3 Searching Strategies

Searching has been deeply studied in Artificial Intelligence. In its basic form, a *state* is a description of the world and two states are linked by a *transition* which allows to reach a state from a previous one. In this way, a directed graph where the nodes are the states and the edges are the actions is considered. Given a starting state, a sequence of actions 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 in the topology of the graph and the order in which new nodes are reached.

The two basic blind search strategies are depth-first search and breadth-first search. The main difference between them is that depth-first search follows a path to its completion before trying an alternative path. Some paths can be infinite, so this search may never succeed. It involves *backtracking*: 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 nodes in the search tree. The complexity in time is linear on the maximum of the number of vertices and the number of edges and the complexity in space is quadratic. In breadth-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. In this case the complexity in time and in space is the same as for depth-first search.

4 Depth-First Search with P Systems

The idea of representing an instantaneous description of the world as a state and a transition 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, a first approach to depth-first search with P systems is presented. The aim of this first approach is not to provide a minimalist approach. We are not looking for the minimum number of ingredients for implementing the depth-first search in P systems. 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, a representation of a problem $P = (a, S, E, F)$ as a space of states consists of 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 and a set F of final states. Technically, a *cost* mapping is also needed, 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 $\Pi = (\Gamma, H, \mu, w_u, w_s, R_1, R_2, R_3, R_1 > R_2 > R_3)$ where

- The alphabet $\Gamma = S \cup \{p_x \,|\, x \in S\} \cup \{r_e \,|\, e \in E\}$
- The set of labels $H = \{u, s\}$
- A membrane structure $\mu = [\,[\,]_u\,]_s$
- The initial multisets $w_u = \{a\}$ and $w_s = \emptyset$.
- The sets of rules R_1, R_2 and R_3 are associated with the membrane u:
  - $R_1 = \{[x]_u \to \lambda : x \in F\}$. For each final state we have a dissolution rule which dissolves the membrane u.
  - $R_2 = \{[x \neg p_y \to y\, r_{xy}]_u : (x, y) \in E\}$. For each transition (x, y), x produces $y\, r_{xy}$ if p_y does not occur in the membrane u, i.e., p_y acts as an inhibitor.
  - $R_3 = \{[y\, r_{xy} \to x\, p_y]_u : (x, y) \in E\}$. For each transition (x, y) we have a cooperative rule where the multiset $y\, r_{xy}$ is rewritten as $x\, p_y$ in the membrane u.
- An order among the rules is considered. Rules of R_1 have higher priority than the other rules and rules from R_2 have priority over rules from R_3.

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[1] which forbids 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.

4.1 Example

Let us consider a representation of a problem as a space of states $P = (a, S, E, F)$ with $S = \{a, b, c, d, e, f, g\}$, a the initial state, the set of transitions $E = \{(a, b),$

[1] Notice that 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.

$(a,c),(b,d),(b,e),(e,f),(c,g)\}$ 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 from 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 $C_1 = [\,[b\,r_{ab}]_u\,]_s$ is obtained. From C_1, three rules are applicable $r_d \equiv [b\,\neg p_d \rightarrow d\,r_{bd}]_u \in R_2$, $r_e \equiv [b\,\neg p_e \rightarrow e\,r_{be}]_u \in R_2$ and $r_{\underline{b}} \equiv [b\,r_{ab} \rightarrow a\,p_b]_u \in R_3$.

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, $r_f \equiv [e\,\neg p_f \rightarrow f\,r_{ef}]_u \in R_2$ and $r_{\underline{e}} \equiv [e\,r_{be} \rightarrow b\,p_e]_u \in R_3$. Since R_2 has priority, r_f is applied and the configuration $C_3 \equiv [\,[f\,r_{ab}\,r_{be}\,r_{ef}]_u\,]_s$ is reached. 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 backtracking. 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 backtracking 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, $r_d \equiv [b\,\neg p_d \rightarrow d\,r_{bd}]_u \in R_2$ and $r_{\underline{b}} \equiv [b\,r_{ab} \rightarrow a\,p_e]_u \in R_3$. 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 and the configuration $C_6 \equiv [\,[d\,r_{ab}\,r_{bd}\,p_e\,p_f]_u\,]_s$ is reached. From C_6 only backtracking can be done by applying the rule $r_{\underline{d}} \equiv [d\,r_{bd} \rightarrow b\,p_d]_u \in R_3$ and reach $C_7 \equiv [\,[b\,r_{ab}\,p_d\,p_e\,p_f]_u\,]_s$. By applying now $r_{\underline{b}} \equiv [b\,r_{ab} \rightarrow a\,p_b]_u \in R_3$ the configuration $C_8 \equiv [\,[a\,p_b\,p_d\,p_e\,p_f]_u\,]_s$ is obtained. From C_8 we only can apply $r_c \equiv [a\,\neg p_c \rightarrow c\,r_{ac}]_u \in R_2$ and reach $C_9 \equiv [\,[c\,r_{ac}\,p_b\,p_d\,p_e\,p_f]_u\,]_s$. From C_9 two rules are applicable, $r_g \equiv [c\,\neg p_g \rightarrow g\,r_{cg}]_u \in R_2$ and $r_{\underline{c}} \equiv [c\,r_{ac} \rightarrow a\,p_c]_u \in R_3$. Due to the priority of R_2 over R_3, r_g is applied and the configuration $C_{10} \equiv [\,[g\,r_{ac}\,r_{cg}\,p_b\,p_d\,p_e\,p_f]_u\,]_s$ is obtained. Finally, the applicable rules are $r_F \equiv [g]_u \rightarrow \lambda \in R_1$ and $r_{\underline{g}} \equiv [g\,r_{cg} \rightarrow c\,p_g]_u \in R_3$. Since R_1 has priority over R_3, the rule r_F is applied and the configuration $C_{11} \equiv [r_{ac}\,r_{cg}\,p_b\,p_d\,p_e\,p_f]_s$. No more rules are applicable and C_{11} 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. We have chosen an *incremental formulation* (see [8]), which starts from the empty state and each action adds a queen to the state. This 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 this way, states are arrangements of k queens ($0 \leq k \leq N$), one per column in the leftmost k columns and transitions are pairs (x,y) where the state

y is the state x with a new queen is added in the leftmost empty column. Such new queen is not attacked by any other one already present on the board.

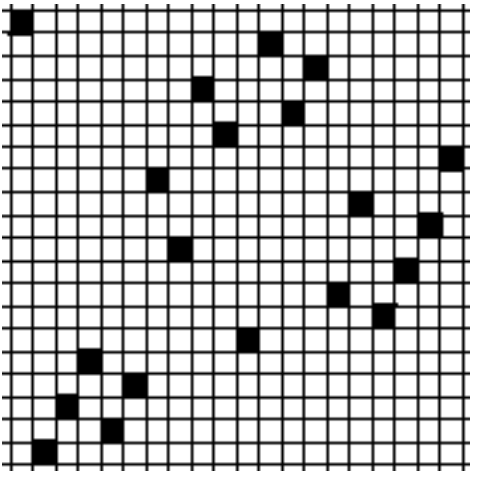
The basic idea of the P system design is to encode the position of a queen as a set of four objects x_i, y_j, u_{i-j} and v_{i+j}, where x_i represents a column and y_j represents a row ($1 \leq i, j \leq N$). The objects u_{i-j} and v_{i+j} represent the ascendant and the descendant diagonals respectively 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 then the process finishes; otherwise we do backtracking and choose another eligible set.

We present a family of P systems which solves the decision problem associated to 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 removing useless objects. For each positive integer greater than 2, we consider the P system $\Pi = (\Gamma, H, \mu, w_u, w_s, R_1, R^*, R_2, R_3, R_1 > R^* > R_2 > R_3)$ 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
  - $R_1 = \{[x_{N+1}]_u \rightarrow \lambda : x \in F\}$. In this design, when the object k_N is reached, the membrane u is dissolved and the computation ends.
  - $R^* = \{[p_{i,j}x_{i-1} \rightarrow x_{i-1}]_u : i \in \{2, \ldots, N\}, j \in \{1, \ldots, N\}\}$ Just cleaning rules.
  - $R_2 = \{[x_i\, y_j\, u_{i-j}\, v_{i+j}\, \neg p_{i,j} \rightarrow x_{i+1}\, r_{i,j}\,]_u : i, j \in \{1, \ldots, N\}\}$ These rules put a new queen on the chessboard by choosing an eligible position.
  - $R_3 = \{[r_{i,j}\, x_{i+1} \rightarrow x_i\, y_j\, u_{i-j}\, v_{i+j}\, p_{i,j}]_u : i, j \in \{1, \ldots, N\}\}$. These rules remove one queen from the chessboard and implement the backtracking.
- Finally, the order $R_1 > R^* > R_2 > R_3$ among the sets of rules is settled.

5.1 A Brief Overview of 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 $[k_0\, x_1\, y_j\, u_{1-j}\, v_{1+j}\, \neg p_{1,j,0} \rightarrow x_2\, r_{1,j,1}\, k_2]_u$ where $j \in \{1, \ldots, N\}$ is chosen. The choice 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 not eligible and the new queen will be placed in a *safe* square, in the sense that no other queen in the board threatens this position (i.e., there are no other queens in the same row, nor in the same column, nor in both diagonals). The application of the rule produces the object x_2. Next, a rule from the set



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. 1. A solution for the 20-queens problem

$[k_1\, x_2\, y_j\, u_{2-j}\, v_{2+j}\, \neg p_{2,j,1} \rightarrow x_3\, r_{2,j,2}\, k_3]_u$ 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 backtracking and produce 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

An *ad hoc* CLIPS program (available from the authors) has been written based on this design of solution for the N-queens problem based on Membrane Computing techniques. Some experiments have been performed on a system with an Intel Pentium Dual CPU E2200 at 2,20 GHz, 3GB of RAM and using CLIPS V6.241 under the operating system Windows Vista. Finding one solution took 0,062 seconds for a 4×4 board and 15,944 seconds for a 20×20 board. Figure 1 shows a solution for the 20-queens problem found by this computer program.

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 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 simpler P system models. The second improvement is associated to the nature of P systems. The design of P systems which compute 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 use a limited form of parallelism where several rules can be applied simultaneously, but controlling the exponential explosion of brute force algorithms. The current parallel computing architectures (see, e.g., [4]) can be a clue for these new generations of *membrane 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. Díaz-Pernil, D., Pérez-Hurtado, I., Pérez-Jiménez, M.J., Riscos-Núñez, A.: A P-lingua programming environment for membrane computing. In: Corne, D.W., Frisco, P., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2008. LNCS, vol. 5391, pp. 187–203. Springer, Heidelberg (2009)
2. Gutiérrez-Naranjo, M.A., Martínez-del-Amor, M.A., Pérez-Hurtado, I., Pérez-Jiménez, M.J.: Solving the n-queens puzzle with P systems. In: Gutiérrez-Escudero, R., et al. (eds.) Seventh Brainstorming Week on Membrane Computing, Fénix Editora, Sevilla, Spain, vol. I, pp. 199–210 (2009)
3. Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Riscos-Núñez, A., Romero-Campero, F.J.: Computational efficiency of dissolution rules in membrane systems. International Journal of Computer Mathematics 83(7), 593–611 (2006)
4. Martínez-del-Amor, M.A., Pérez-Hurtado, I., Pérez-Jiménez, M.J., Cecilia, J.M., Guerrero, G.D., García, J.M.: Simulation of recognizer P systems by using manycore gpus. In: Martínez-del-Amor, M.A., et al. (eds.) Seventh Brainstorming Week on Membrane Computing, Fénix Editora, Sevilla, Spain, vol. II, pp. 45–58 (2009)
5. Păun, G., Rozenberg, G., Salomaa, A. (eds.): Handbook of Membrane Computing. Oxford Univ. Press, Oxford (2010)
6. Pérez-Jiménez, M.J.: A computational complexity theory in membrane computing. In: Păun, G., Pérez-Jiménez, M.J., Riscos-Núñez, A., Rozenberg, G., Salomaa, A. (eds.) WMC 2009. LNCS, vol. 5957, pp. 125–148. Springer, Heidelberg (2010)
7. Pérez-Jiménez, M.J., Romero-Jiménez, Á., Sancho-Caparrini, F.: Complexity classes in models of cellular computing with membranes. Natural Computing 2(3), 265–285 (2003)
8. Russell, S.J., Norvig, P.: Artificial Intelligence: A Modern Approach, 2nd edn. Prentice-Hall, Englewood Cliffs (2002)
9. Wikipedia, `http://en.wikipedia.org/wiki/orders_of_magnitude_mass`