Depth-first Search with P Systems

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

Research Group on Natural Computing

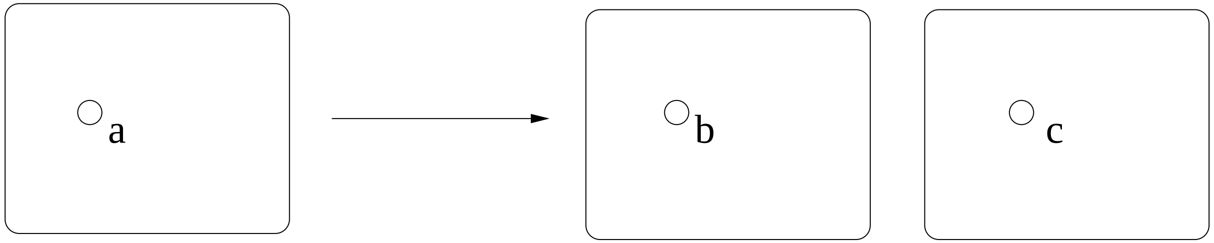
Dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla

▷ **Many problems:**

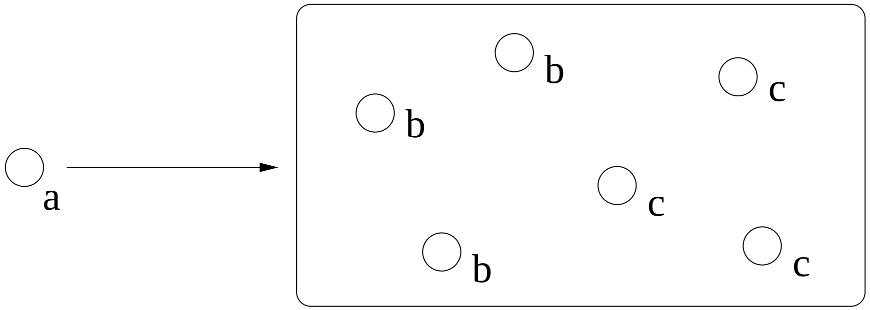
- **SAT: The problem of propositional satisfiability for formulas in conjunctive normal form**
- **Subset Sum: Given a finite set, A, a weight function, $w : A \to N$, and a constant $k \in \mathbb{N}$, determine whether or not there exists a subset $B \subseteq A$ such that $w(B) = k$. If A has n elements with weights $w_1, \ldots, w_n$, one instance of the problem can be encoded as $(n, (w_1, \ldots, w_n), k)$.**
- **Partition: Given a set $A = \{a_1, \ldots, a_n\}$, where each element a_i has a** *weight* **$w_i \in \mathbb{N}$, decide whether or not there exists a partition of A into two subsets such that they have the same weight.**
- **...**

New membranes

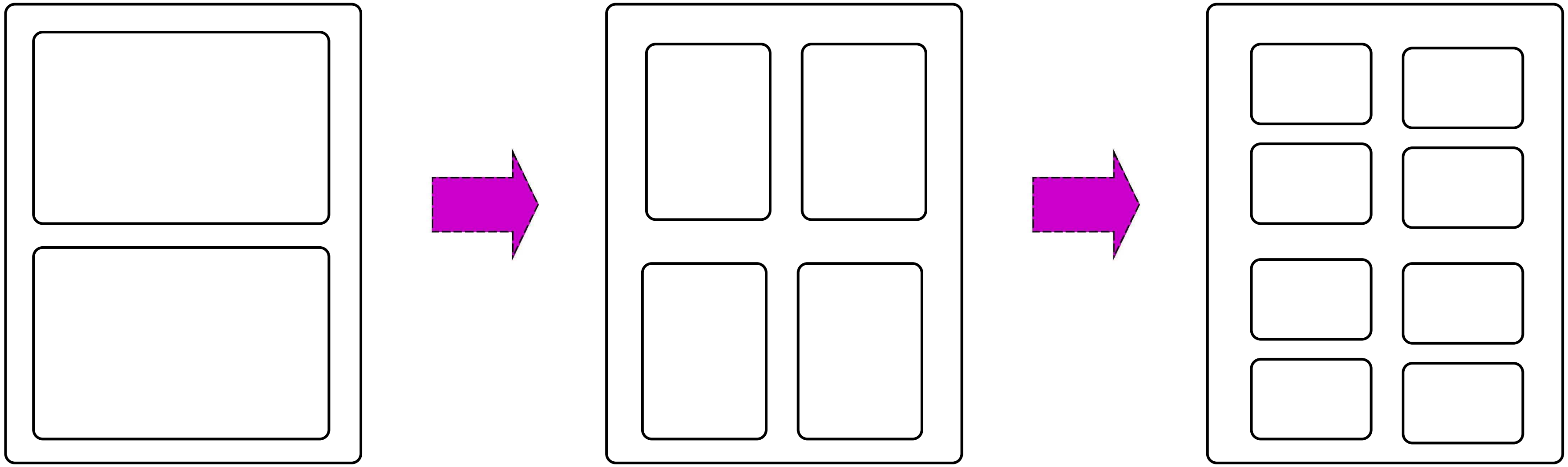
▷ **Divison of membranes** *(based on mitosis)*: **P Systems with Active Membranes**



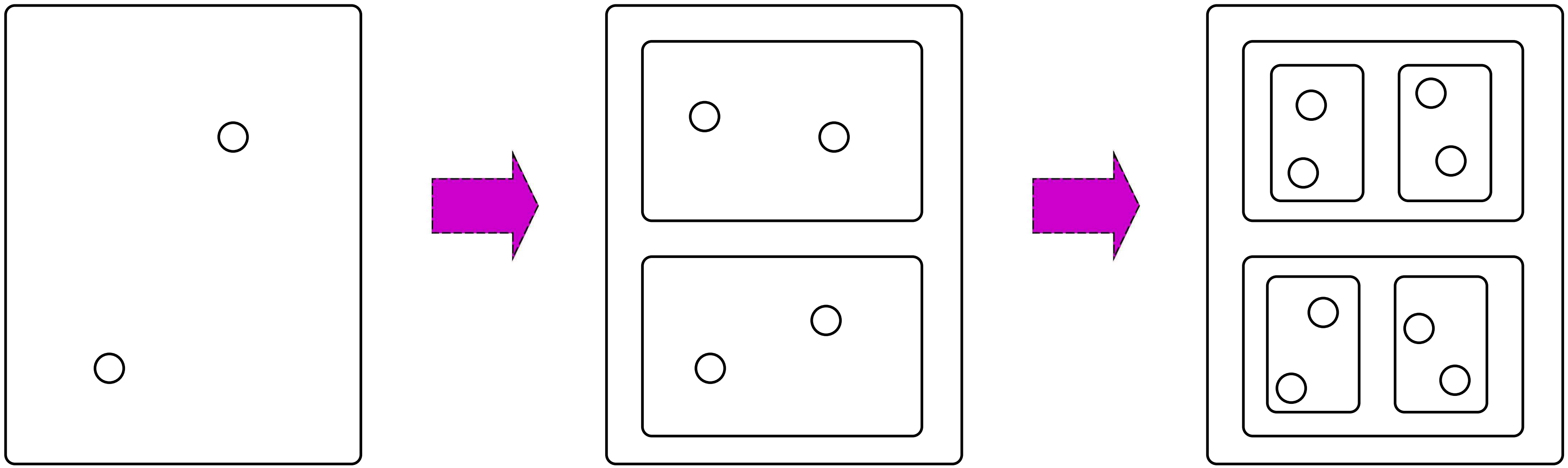
▷ **Creation of membranes** *(based on autopoiesis)*: **P Systems with Membrane Creation**



▷ **Division of membranes:**



▷ **Creation of membranes:**



Caveat!

- **Generating an exponential amount of membranes in linear time may not be enough!**

Caveat!

▷ **Generating an exponential amount of membranes in linear time may not be enough!**

▷ **Păun's Conjecture:**

- **Object evolution rules**
- **Communication rules (send-in and send-out)**
- **Division of membranes**
- **Dissolution of membranes**
- **Without polarizations**

Caveat!

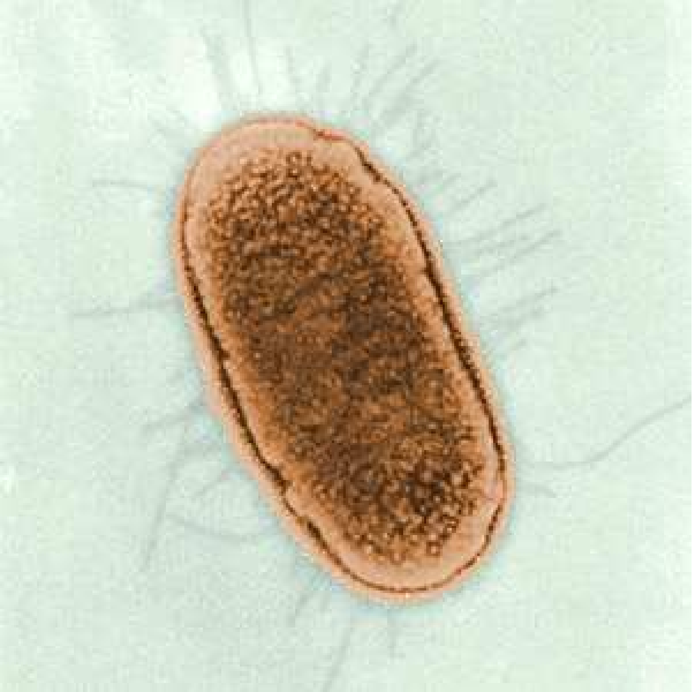
- ▷ **Generating an exponential amount of membranes in linear time may not be enough!**
- ▷ **Păun's Conjecture:**
  - **Object evolution rules**
  - **Communication rules (send-in and send-out)**
  - **Division of membranes**
  - **Dissolution of membranes**
  - **Without polarizations**
- ▷ **Other ingredients: Priorities, cooperation, electrical charges, . . .**
- ▷ **Several semantics**

A common scheme

▷ **A common scheme:**

- **Generation stage: membrane division or membrane creation is used to bulid an exponential amount of membranes.**
- **Calculation stage: in each membrane, a feasible candidate solution is encoded.**
- **Checking stage: in each membrane it is checked if the candidate is a solution**
- **Output stage: The results of the checkin is collected and a final answer is delivered.**

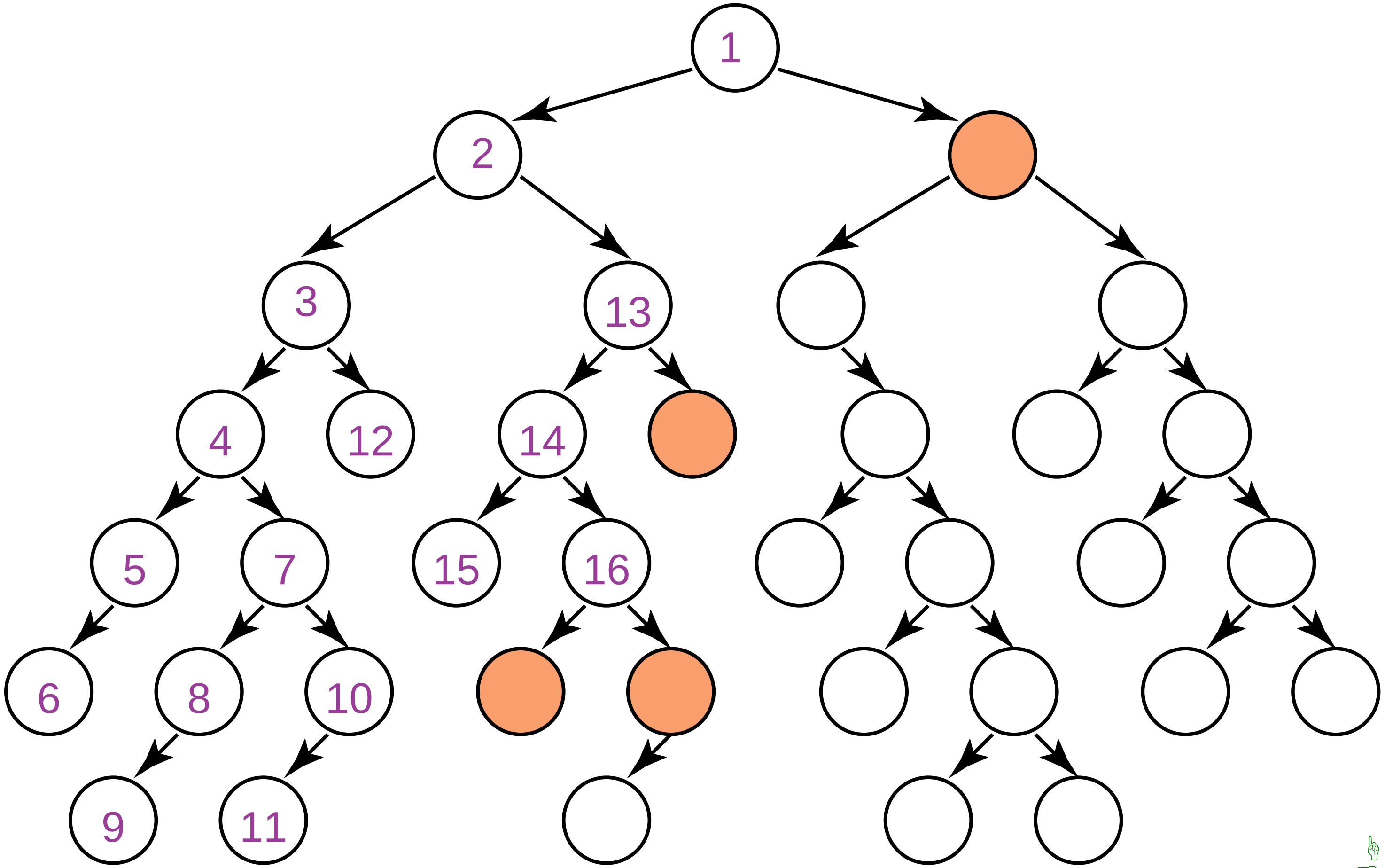
- **In vivo implementation where each feasible solution is encoded in an elementary membrane**
- **Such elementary membrane is** *implemented* **in a bacteria of mass similar to E. Coli (**$\sim 7 \times 10^{-16}$ **kg.),**
- **An instance of a NP problem with input size 40 will need approximately the mass of the Earth for an implementation (**$\sim 6 \times 10^{24}$ **kg.)**



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.**
- ▷ **The order in which the nodes are explored determines the searching strategy**

Depth-first Search



Depth-first Search

▷ **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.**

Ingredients?

- Dissolution
- Cooperation
- Inhibitors
- Priority

Ingredients?

- **Dissolution**
- **Cooperation**
- **Inhibitors**
- **Priority**

- **Open problem:**
  Remove ingredients

- **P system** $\Pi = (\Gamma, H, \mu, w_e, w_s, R_1, R_2, R_3, R_1 > R_2 > R_3)$ **with priorities**

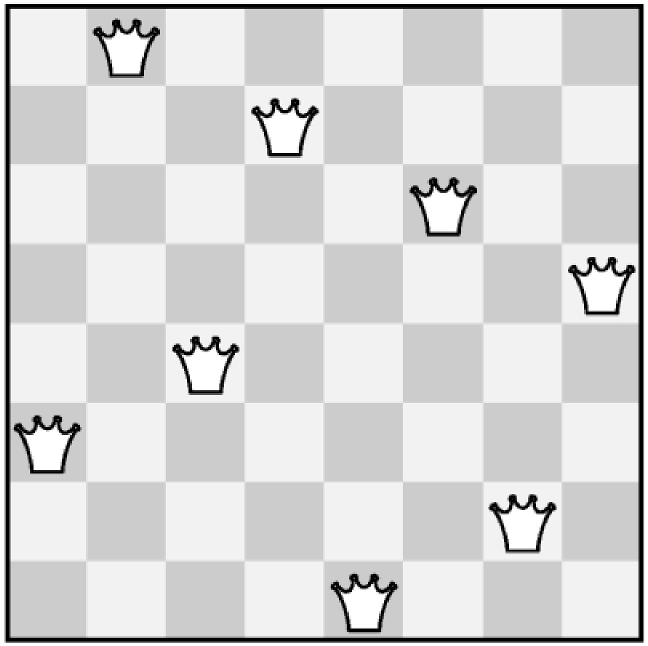
- **The alphabet** $\Gamma = S \cup \{p_e, r_e \,|\, e \in E\}$**, the set of labels** $H = \{u, s\}$**, the membrane structure** $\mu = [\,[\,]_u\,]_s$**, the initial multisets** $w_u = \{a\}$ **and** $w_s = \emptyset$ **and the sets of rules** R_1**,** R_2 **and** R_3
  - $R_1 = \{[x]_u \to \lambda : x \in F\}$**. 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 **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.**
  - $R_3 = \{[y\,r_{xy} \to x\,p_y]_u : (x, y) \in E\}$**. 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**.**

▷ **Intuition behind the objects is the following:**

- **Recall the current state (one object from S) in the configuration. It represents the current state in the searching process.**
- **Recall the forbidden nodes (objects p_y)**
- **Recall the path to the current node (objects r_{xy})**

A Case Study: The N-queens puzzle

- **The N-queens puzzle consists on placing N pieces (queens) on an N$\times$N grid in such way that no two queens are on the same row, column or diagonal line.**



The N-queens puzzle

The N-queens puzzle ⇒ A constraint problem ⇒ A formula in CNF ⇒ P system

The N-queens puzzle

- There is at most one queen in each column.
- There is at most one queen in each row.
- There is at most one queen in each ascendant diagonal line.
- There is at most one queen in each descendant diagonal line.
- There is at least one queen in each column.

A formula in CNF

- $\psi_1 \equiv \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{n} \bigwedge_{k=j+1}^{n} (\neg s_{ij} \vee \neg s_{ik})$ $\qquad \psi_2 \equiv \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{n} \bigwedge_{k=j+1}^{n} (\neg s_{ji} \vee \neg s_{ki})$

- $\psi_3 \equiv \bigwedge_{d=0}^{n-2} \bigwedge_{j=1}^{n-d} \bigwedge_{k=j+1}^{n-2} (\neg s_{d+j\,j} \vee \neg s_{d+k\,k})$ $\quad \psi_4 \equiv \bigwedge_{d=-(n-2)}^{-1} \bigwedge_{j=1}^{n+d} \bigwedge_{k=j+1}^{n+d} (\neg s_{j\,j-d} \vee \neg s_{k\,k-d})$

- $\psi_5 \equiv \bigwedge_{d=3}^{n+1} \bigwedge_{j=1}^{d-1} \bigwedge_{k=j+1}^{d-1} (\neg s_{j\,d-j} \vee \neg s_{k\,d-k})$ $\quad \psi_6 \equiv \bigwedge_{d=n+2}^{2n-1} \bigwedge_{j=d-n}^{n} \bigwedge_{k=j+1}^{d-1} (\neg s_{j\,d-j} \vee \neg s_{k\,d-k})$

- $\psi_7 \equiv \bigwedge_{i=1}^{n} \bigvee_{j=1}^{n} s_{ij}$

- $$\Phi \equiv \psi_1 \bigwedge \psi_2 \bigwedge \psi_3 \bigwedge \psi_4 \bigwedge \psi_5 \bigwedge \psi_6 \bigwedge \psi_7$$

P system (rules)

$(a,1)$ $[\, d_j \,]_2^0 \rightarrow [\, s_{j+1} \,]_2^+ [\, s_{j+1} \,]_2^-$ for all $j \in \{0, \ldots, n-1\}$.

$(a,2)$ $[\, d_j \,]_2^+ \rightarrow d_j \, [\,]_2^0 \quad [\, d_j \,]_2^- \rightarrow d_j \, [\,]_2^0$ for all $j \in \{1, \ldots, n\}$.

$(a,3)$ $d_j \, [\,]_2^0 \rightarrow [\, d_j \,]_2^0$ for all $j \in \{1, \ldots, n-1\}$.

$(a,4)$ $[\, d_i \rightarrow d_{i+1}]_1^0$ for all $i \in \{n, \ldots, 3n-4\} \cup \{3n-2, \ldots, 3n+2m\}$.

$(a,5)$ $[\, d_{3n-3} \rightarrow d_{3n-2} e]_1^0$.

$(a,6)$ $[\, d_{3n+2m+1} \,]_1^0 \rightarrow \mathtt{No} \, [\,]_1^+$.

(b) $[\, s_j \rightarrow t_j d_j \,]_2^+ \quad [\, s_j \rightarrow f_j d_j \,]_2^-$ for all $j \in \{1, \ldots, n\}$.

$(c,1)$ $\left\{ \begin{array}{ll} [\, x_{i1} \rightarrow r_{i1} \,]_2^+ & [\, y_{i1} \rightarrow \lambda \,]_2^+ \\ [\, x_{i1} \rightarrow \lambda \,]_2^- & [\, y_{i1} \rightarrow r_{i1} \,]_2^- \end{array} \right\}$ for all $i \in \{1, \ldots, m\}$.

$(c,2)$ $\left\{ \begin{array}{ll} [\, x_{ij} \rightarrow z_{ij} \,]_2^+ & [\, y_{ij} \rightarrow h_{ij} \,]_2^+ \\ [\, x_{ij} \rightarrow z_{ij} \,]_2^- & [\, y_{ij} \rightarrow h_{ij} \,]_2^- \end{array} \right\}$ for all $i \in \{1, \ldots, m\}$ and $j \in \{2, \ldots, n\}$.

(d) $\ldots$

▷ **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., Gutiérrez-Naranjo, M.A., Păun, Gh., Pérez-Hurtado, I., Riscos-Núñez, A. (eds.) Seventh Brainstorming Week on Membrane Computing. vol. I, pp. 199–210, Fénix Editora, Sevilla, (2009)**

Experimental results: 4-queens puzzle

- **Simulation: P-lingua simulator** *http://www.p-lingua.org*
- **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.**
- **A formula in CNF with 16 variables and 80 clauses.**
- **The** *input multiset* **has 168 elements.**
- $2^{16} = 65536$ **elementary membranes need to be considered in parallel**
- **20587 seconds (**> 5 **hours).**

Experimental results: 4-queens puzzle

▷ **The answer** `Yes`

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

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

13	X	15	16
9	10	11	X
X	6	7	8
1	2	X	4

13	14	X	16
X	10	11	12
5	6	7	X
1	X	3	4

The 5-queens puzzle

- **The 5-queens puzzle needs $2^{25} = 33554432$ simultaneous elementary membranes**
- **It is impossible to deal with so many membranes with current simulators.**
- **What about using a P system implementing depth-first rearch?**
- **Can we find one solution to the N-queens puzzle?**

- **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.**

- **Codification: 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.**

- **Set of rules:**
  - $R_1 = \{[x_{N+1}]_u \rightarrow \lambda : x \in F\}$. **In this design, when the object k_N is reached, the membrane u is dissolved and the computation ends.**
  - $R^* = \{[p_{i,j} x_{i-1} \rightarrow x_{i-1}]_u : i \in \{2, \dots, N\},\, j \in \{1, \dots, N\}\}$ **Deleting useless objects.**
  - $R_2 = \{[x_i\, y_j\, u_{i-j}\, v_{i+j} \neg p_{i,j} \rightarrow x_{i+1}\, r_{i,j}\,]_u : i, j \in \{1, \dots, N\}\}$ **These rules put a new queen on the chessboard by choosing an eligible position.**
  - $R_3 = \{[r_{i,j}\, x_{i+1} \rightarrow x_i\, y_j\, u_{i-j}\, v_{i+j}\, p_{i,j}]_u\ i, j \in \{1, \dots, N\}\}$. **These rules remove on queen form the chessboard and implement the backtracing.**

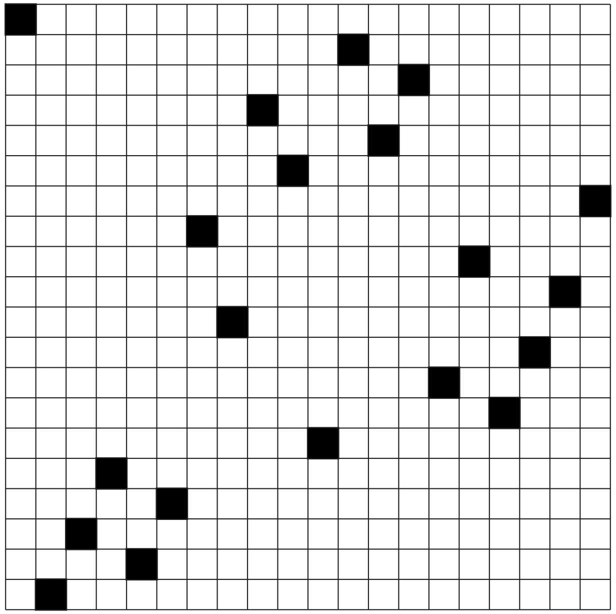
Experiments

- **An *ad hoc* CLIPS simulator has been written**
  - **Intel Pentium Dual CPU E2200 at 2,20 GHz, 3GB of RAM**
  - **CLIPS V6.241 under Windows Vista**

- **It took 0,062 seconds for a 4×4 board**
- **It took 15,944 seconds for a 20×20 board.**

Example

▷ **A solution for the 20-queens problem found by the *ad hoc* CLIPS simulator**



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**

Final remarks

- **Brute force algorithms have been widely used in the design of solutions for many problems in Membrane Computing.**
- **Division and creation of membranes allow us to have as many membranes as we need.**
- **The usual idea: Generate all feasible solutions and check.**

Final remarks

- ▷ **Brute force algorithms have been widely used in the design of solutions for many problems in Membrane Computing.**
- ▷ **Division and creation of membranes allow us to have as many membranes as we need.**
- ▷ **The usual idea: Generate all feasible solutions and check.**
- ▷ **But, if we want effective solutions we need to leave brute force and explore new paths**
- ▷ **Heuristics?**

Final remarks

- **Brute force algorithms have been widely used in the design of solutions for many problems in Membrane Computing.**
- **Division and creation of membranes allow us to have as many membranes as we need.**
- **The usual idea: Generate all feasible solutions and check.**
- **But, if we want effective solutions we need to leave brute force and explore new paths**
- **Heuristics?**

Thanks!