A Cellular Sudoku Solver

Daniel Díaz-Pernil, Carlos M. Fernández-Márquez, Manuel García-Quismondo, Miguel A. Gutiérrez-Naranjo, Miguel A. Martínez-del-Amor

Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012, Sevilla, Spain
sbdani@us.es, carfermar@alum.us.es, mangarfer2@alum.us.es, magutier@us.es, mdelamor@us.es

Summary. Sudoku is a very popular puzzle which consists on placing several numbers in a squared grid according to some simple rules. In this paper we present an efficient family of P systems which solve sudoku puzzles of any order verifying a specific property. The solution is searched by using a simple human-style method. If the sudoku cannot be solved by using this strategy, the P system detects this drawback and then the computations stops and returns `No`. Otherwise, the P system encodes the solution and returns `Yes` in the last computation step.

1 Introduction

Sudoku is currently one of the most famous puzzles in the world. The most popular version consists on a 9×9 grid made up of 3×3 subgrids, but the general case, an $n^2 \times n^2$ grid with $n \times n$ subgrids is considered. Some cells contain numbers, which can be considered as *input data*. The goal is to fill in the empty cells, one number in each, so that each column, row, and subgrid contains the numbers 1 through 9 exactly once (numbers 1 to n^2 in the general case). If the input data are correct, the sudoku has one and only one solution.

The creator is believed to be Howard Garns [6]. He is likely to be the inventor of a puzzle called "Number Place" that appeared in New York in 1979. The puzzle was introduced in Japan by the publishing company Nikoli in the paper *Monthly Nikolist* in April 1984 as *Suji wa dokushin ni kagiru* [7], which can be translated as *the numbers must occur only once* or *the numbers must be single*. Later, the name was abbreviated as *sudoku*, where *su* stands for *number* and *doku* stands for *alone*. Later Wayne Gould from New Zealand discovered the puzzle on a trip to Japan and wrote a program to generate new puzzles. He convinced The Times of London to publish Sudoku puzzles in 2004.

In addition to its undoubted success in entertainment, sudoku has important properties from a mathematical point of view. The first natural question is to won-

der about is the number of all possible sudoku grids. The answer to this question is not an easy matter. A valid sudoku solution is also a Latin square. A Latin square is an $n \times n$ table filled by using numbers from 1 to n in such way that each symbol occurs exactly once in each row and exactly once in each column. The number of 9×9 Latin squares is about 5.525×10^{27}.

Sudoku imposes the additional constraint on subgrids, so from the previous number we need to remove the Latin squares which do not satisfy the condition. The number of valid sudoku solution grids for the standard 9×9 grid is 6,670,903,752,021,072,936,960. This number is equal to $9! \times 72^2 \times 2^7 \times 27,704,267,971$, the last factor of which is prime. The result was derived through logic and brute force computation. The details can be found at [1]. Other important property is that it has been proved that the general problem of solving sudoku puzzles on $n^2 \times n^2$ grids of $n \times n$ boxes is known to be **NP**-complete [5].

Nonetheless, the number of possible solution grids is not the object of study of this paper nor the complexity of finding the solution. In this paper we study the problem of solving sudoku by using Membrane Computing techniques. In the first part of the paper, we develop a theoretical study about the use of brute force algorithms to solve it, based on a well-known solution for the SAT problem. As we will see below, the number of elementary membranes for a usual 9×9 sudoku exceeds the number of atoms of the observable universe, so we have a good reason for looking for a different strategy.

In the second part of the paper, we present a family of P systems $\{\Pi(n)\}_{n\in\mathbb{N}}$ such that $\Pi(n)$ is a P system with input. Such an input is, of course, the input for one sudoku puzzle encoded as a multiset. The solution is searched by using a human-style method based on looking for squares where only one candidate can be placed. This method is good enough to find the solution for a large amount of sudokus, but not all the sudokus can be solved by using this method. An original control method in the design of the algorithm is that the P system stops if the sudoku cannot be solved, i.e., instead of going into a non-ending search, the P system detects the drawback and halts. If the solution can be reached, the P systems stops, sends out an object `Yes` to the environment and provides the solution encoded on the skin. Otherwise, if the P system detects that the solution cannot be reached then it halts and sends `No` to the environment in the last step of computation.

The paper is organized as follows: first we explore a theoretical brute force algorithm based on a Membrane Computing solution for the SAT problem. After showing the practical drawback of such a solution, we present our efficient family of P systems for solving a large amount of sudokus. We illustrate the behavior of this family with an overview of the computation and, finally, some final remarks are presented.

2 A Brute Force Algorithm

A *sudoku square* of order n consists of n^4 constants (usually n^2 copies of the numbers $1, 2, \ldots, n^2$), arranged into an $n^2 \times n^2$ grid which comprises n^2 subgrids of size $n \times n$ (also called boxes). Such a grid verifies that the entries in each row, each column and each box are all different. A *sudoku problem* consists on a partial assignment of the variables in a Sudoku square. The target is to find a completion of the assignment which extends the partial assignment and satisfies the constraints.

The first idea for solving a sudoku problem is to consider it as a constraint problem. In fact, we are looking for one assignment of numbers to squares which satisfies a finite amount of restrictions. The set of constraints of a sudoku problem can be expressed as a logic formula in conjunctive normal form. Following [3], a sudoku square of order n can be represented as an instance of the SAT problem with n^6 propositional variables. For each entry in the $n^2 \times n^2$ grid, we will consider n^2 variables. Let us use the notation s_{xyz} to refer to variables. Variable s_{xyz} is assigned *true* if and only if the entry in the row x and column y is number z. In this way, if the variable s_{753} takes the value *true*, then it means that the number 3 is placed at position $(7, 5)$ of the grid. According to this notation, the different constraints for the sudoku problem can be represented as the following formulae:

- There is at least one number in each entry:
$$\psi_1 \equiv \bigwedge_{x=1}^{n^2} \bigwedge_{y=1}^{n^2} \bigvee_{z=1}^{n^2} s_{xyz}$$
- Each number appears at most once in each row:
$$\psi_2 \equiv \bigwedge_{y=1}^{n^2} \bigwedge_{z=1}^{n^2} \bigwedge_{x=1}^{n^2-1} \bigwedge_{i=x+1}^{n^2} (\neg s_{xyz} \vee \neg s_{iyz})$$
- Each number appears at most once in each column:
$$\psi_3 \equiv \bigwedge_{x=1}^{n^2} \bigwedge_{z=1}^{n^2} \bigwedge_{y=1}^{n^2-1} \bigwedge_{i=y+1}^{n^2} (\neg s_{xyz} \vee \neg s_{xiz})$$
- Each number appears at most one in each box:
$$\psi_4 \equiv \bigwedge_{z=1}^{n^2} \bigwedge_{i=0}^{n-1} \bigwedge_{j=0}^{n-1} \bigwedge_{x=1}^{n} \bigwedge_{y=1}^{n} \bigwedge_{k=y+1}^{n} (\neg s_{(ni+x)(nj+y)z} \vee \neg s_{(ni+x)(nj+k)z})$$
$$\psi_5 \equiv \bigwedge_{z=1}^{n^2} \bigwedge_{i=0}^{n-1} \bigwedge_{j=0}^{n-1} \bigwedge_{x=1}^{n} \bigwedge_{y=1}^{n} \bigwedge_{k=x+1}^{n} \bigwedge_{l=1}^{n} (\neg s_{(ni+x)(nj+y)z} \vee \neg s_{(ni+k)(nj+l)z})$$

The conjunction of these five formulae $\Phi \equiv \psi_1 \bigwedge \psi_2 \bigwedge \psi_3 \bigwedge \psi_4 \bigwedge \psi_5$ is a formula in conjunctive normal form and each truth assignment which makes it true represents a right arrangement of n^2 copies of $1, 2, \ldots, n^2$ according to the sudoku constraints.

Once expressed the sudoku as such a formula, finding a solution to the puzzle can be considered as the problem of finding a truth assignment which satisfies the formula. This is exactly the satisfiability (SAT) problem.

Any truth assignment which makes it true represents a right arrangement of n^2 copies of $1, 2, \ldots, n^2$ according to the sudoku constraints over an empty sudoku grid of order n, but we are not interested in finding such solutions. In fact, a sudoku puzzle should have a certain amount of numbers placed in the right position as input in such way that there exists a unique possible assignment which represents the solution to the problem. Given a sudoku puzzle, we will call *Input* to the set

$$Input = \{\langle x, y, z\rangle : z \text{ is placed on the square } (x, y) \text{ of the grid}\}$$

In order to deal with the input, it is enough to add the corresponding values to the formula Φ as follows:

$$\Phi_{in} \equiv \Phi \wedge \bigwedge_{\langle x,y,z\rangle)\in Input} s_{xyz}$$

Given a sudoku problem and its associated formula Φ_{in}, finding the solution to the problem is equivalent to finding a truth assignment which satisfies the formula.

In [2], a family of P systems with active membranes to solve the SAT problem was presented. It was based on the solution presented in [4]. The main difference was that the solution from [4] only provides `Yes` or `No` as answers to the problem. The solution in [2] found and stored all the truth assignments which satisfy the formula, if there exists.

By considering the encoding of a sudoku problem as a CNF formula on one side and the family of P systems which provides solutions for SAT on the other side, we have a Membrane Computing solution for all sudoku problems.

This nice theoretical solution has an insurmountable obstacle from a practical point of view: The number of elementary membranes in one configuration of one P system from the family reaches 2^N, where N is the number of variables of the CNF formula. For a sudoku of order n, the number of variables is n^6, and so the number of elementary membranes is 2^{n^6}. For a usual sudoku of order 3, 2^{729} elementary membranes are simultaneously handled. Estimates of the matter content of the observable universe indicate that it contains on the order of 10^{80} atoms[1], so the brute force algorithm is only a fine calculus for Membrane Computing theory.

3 A New Solution

In this section we present a family of P systems $\mathbf{P} = \{\Pi(n) : n \in \mathbb{N}\}$ such that $\Pi(n)$ solves sudokus of order n. The P system $\Pi(n)$ receives as input the initial data placed in a sudoku puzzle. It is designed to solve sudokus which satisfy a property which will be described below. If the sudoku satisfies the property, the

[1] http://en.wikipedia.org/wiki/Observable_universe

P system computes the solution, it sends an object `Yes` to the environment in the last step of computation and encodes the solution to the sudoku as a multiset in the skin of the halting configuration. Otherwise, the P system detects that the property is not satisfied and halts by sending an object `No` to the environment in the last step of computation.

The property is the following: *In all partial solutions of the sudoku, there exists at least one square (i, j) with a unique candidate.*

We will call *partial solution* of a sudoku to a sudoku grid where some new numbers have been placed and all of them are in the right position. A number p is a unique candidate for the square (i, j) if for all $q \in \{1, \ldots, n^2\}$, $p \neq q$, q has been previously placed in the same row, the same column or the same box of (i, j). For example, if we consider the sudoku of order 2 of Figure 1, number 4 is a unique candidate for the square $(1, 2)$, since 1 is in the same row, 2 is in the same column and 3 is in the same box.

41	42	43	44
1 31	**2** 32	33	34
3 21	22	23	24
11	12	13	**1** 14

Fig. 1. A sudoku problem of order 2

Many sudokus satisfy this property. It is in the base of many human strategies for solving sudokus. Nonetheless, sometimes it is not enough to solve the sudoku and more sophisticated methods are necessary.

3.1 A Family of P Systems

Next, we present a family $\mathbf{\Pi} = \{\Pi(n)\}_{n \in \mathbb{N}}$ for solving any sudoku of order n verifying the property stated above. Each P system $\Pi(n)$ only depends only on the order n of the sudoku and it does not increase the number of membranes along the computation. The used rules are of the following types:

- *Enzymatic rules:* $[\neg in\ u \underset{cat}{\longrightarrow} v]_e$. The multiset u evolves to the multiset v in the membrane with label e. The rule is applied if in the same membrane the objects from the set *cat* are present (catalysts) and none of the objects from the set *in* are present (inhibitors). The catalysts and the inhibitor are not modified by the application of the rules and *cat*, *in* and v can be empty.
- *Dissolution rules:* $[\,u\,]_e \rightarrow o$. The multiset u causes membrane e to dissolve and produces the object o.

- *Send-out rules:* $[\,a\,]_e \to []_e\, a$. The object a is sent out of the membrane with label e.

As usual, all the rules are applied in parallel and in a maximal manner. In one step, one object of a membrane can be used by only one rule (chosen in a non deterministic way), but any object which can evolve by one rule of any form, should evolve. If a membrane is dissolved, its content is left free in the surrounding region. If there are objects in this membrane which evolve by means of enzymatic rules and a membrane h is dissolved at the same time, then we suppose that first the enzymatic rules are used and then the dissolution is produced. Of course, this process takes only one step. We will also use priorities among sets of rules.

The input will be provided as a set of objects z_{ijn} by denoting that the number n is placed at the square with row i and column j. The initial configuration will also contain information about the box corresponding to each square. In such a way, objects box_{ijk} with $i, j \in \{1, \ldots, n^2\}$ are place in the initial configuration and k is the box corresponding to the square (i, j), i.e., if $i = \alpha n + \beta$ and $j = \gamma n + \delta$ with $\alpha, \gamma \in \{0, \ldots, n-1\}$ and $\beta, \delta \in \{1, \ldots, n\}$ then $k = \alpha n + \gamma + 1$.

The initial configuration also contains the objects f_{ix}, c_{jx}, b_{kx}, sq_{ij} with $i, j, k, x \in \{1, \ldots, n^2\}$. The occurrence of f_{ix} in the configuration denotes that the number x is not placed in any square of the row i yet and then, the number x can be eventually placed in such a row in the future. Analogously, c_{jx} denotes that x is not placed in the column j and b_{kx} that the object is not placed in any square of the box k. The objects sq_{ij} are witnesses of the existence of the corresponding square. Finally, n^2 copies of each object r_{ij} with $i, j \in \{1, \ldots, n^2\}$ are also placed in the initial configuration.

The idea of the design is to develop a sequence of two stages: The *checking stage* and the *reset stage*. In the checking stage, the P system looks for squares with a unique candidate. If such squares are found, the candidates are placed in them. After the stage all the auxiliary objects are recalculated in the *reset stage* and then we start again the *checking stage*. This *checking-reset* cycle ends when all the squares are filled and the sudoku is solved or if in a checking stage no new squares with unique candidates are found.

Formally, the P system of order n with input that solves the sudokus with the property claimed above is a construct

$$\Pi(n) = \langle \Gamma, H, \mu, w_e, w_s, i_0, R_1, R_2, R_3^a, R_3^m, R_3^d, R_4, \ldots, R_{11}, R_{12}^1, R_{12}^2 \rangle$$

with the priorities $R_1 > R_2 > R_3^q > R_4 > \ldots R_{11} > R_{12}^x$ with $q \in \{a, r, m\}$ and $x \in \{1, 2\}$ where

- The alphabet $\Gamma = \{s_{ijx}, z_{ijx}, box_{ijk}, sq_{ij}, f_{ix}, c_{jx}, b_{kx}, a_{ij}, r_{ij} \ : \ i, j, k, x \in \{1, \ldots, n^2\}\} \cup \{k_0, k_1, w\}$
- The set of labels $H = \{e, s\}$
- The membrane structure $\mu = [\,[\,]_e\,]_s$
- The initial multisets $w_e = \{k_1\} \cup \{box_{ijk}, sq_{ij}, f_{ix}, c_{jx}, b_{kx}, r_{ij}^{n^2} \ : \ i, j, k, x \in \{1, \ldots, n^2\}\}$ and $w_s = \emptyset$.

- $i_0 = e$, i.e., the input membrane is e.

We also consider the following sets of rules[2]

$$R_1 \equiv [\, z_{ijx} \to s_{ijx}\, p\,]_e \text{ for } i,j,x \in \{1,\ldots,n^2\}.$$

Each input object z_{ijx} produces an object s_{ijx} and one object p. In any configuration we will have as many objects p as numbers are correctly placed on the sudoku. After applying these rules, we have as many objects p as numbers are placed as input.

$$R_2 \equiv \left\{ \begin{array}{l} [f_{ix} \xrightarrow[s_{ijx}]{} \lambda]_e \\ [c_{jx} \xrightarrow[s_{ijx}]{} \lambda]_e \\ [b_{kx} \xrightarrow[s_{ijx}\, box_{ijk}]{} \lambda]_e \end{array} \right\} \text{ for } i,j,k,x \in \{1,\ldots,n^2\}.$$

The object s_{ijx} represents that the number x is placed in the square (i,j). When such an object is generated the objects f_{ix}, c_{jx} and b_{kx} must disappear.

$$\left. \begin{array}{l} R_3^m \equiv [\neg d\, m_{ijx} \xrightarrow[k_1]{} \lambda]_e \\ R_3^a \equiv [\neg d\, a_{ij} \xrightarrow[k_1]{} \lambda]_e \\ R_3^r \equiv [\neg d\, r_{ij} \xrightarrow[k_1]{} \lambda]_e \end{array} \right\} \text{ for } i,j,x \in \{1,\ldots,n^2\}.$$

$$R_4 \equiv [\neg d\, \neg r_{ij} sq_{ij} \xrightarrow[k_1]{} sq_{ij}\, r_{ij}^{n^2}\, d]_e \text{ for } i,j \in \{1,\ldots,n^2\}.$$

Before starting with the checking stage, we ensure that the markers (m_{ijx}) and counters (a_{ij} and r_{ij}) are reset. First, we remove all the copies of a_{ij}, m_{ij} and r_{ij}. In the next step we add n^2 copies of each object r_{ij} to membrane e.

$$R_5 \equiv [k_1 \to k_0]_e$$

The reset stage ends when object k_1 (used as catalyst in the previous sets of rules) evolves to k_0.

$$R_6 \equiv [\neg m_{ijx}\, r_{ij} \xrightarrow[f_{ix}\, c_{jx}\, b_{kx}\, box_{ijk}]{} m_{ijx}\, a_{ij}]_e \text{ for } i,j,x \in \{1,\ldots,n^2\}.$$

The checking stage starts with the set R_6. We know that if objects $f_{ix}\, c_{jx}\, b_{kx}$ are present in membrane e in one configuration, then the number x is a candidate to be placed in the square (i,j), since x has not been placed yet in the row i, the column j or the box k. The question is to know if x is the *unique* candidate. This is checked by rules from set R_6. Before applying these rules, we have checked that for all square (i,j) we have n^2 copies of r_{ij} in membrane e and zero copies of a_{ij}. If $f_{ix}\, c_{jx}\, b_{kx}$ are present in the membrane (they act as catalyst), then the

[2] We write $\neg$ before the object a if a acts as an inhibitor.

corresponding rule is applied. The application of the rule removes one copy of r_{ij} and produces one copy of a_{ij}. The occurrence of m_{ijx} ensures that the rule is applied once.

$$R_7 \equiv [\neg s_{ijq}\, r_{ij}^{n^2-1}\, a_{ij} f_{ix}\, c_{jx}\, b_{kx} \xrightarrow[d\, box_{ijk}]{} s_{ijx}\, p\, w]_e \;\; i,j,x,q \in \{1,\ldots,n^2\}.$$

If there exists only one a_{ij} and n^2-1 copies of r_{ij} (and the square (i,j) is empty) then the square (i,j) has a unique candidate. The rules delete the objects $f_{ix}\, c_{jx}\, b_{kx}$ and introduce the objects s_{ijx}, p and w. The object s_{ijx} corresponds to a number and a square in the solution of the sudoku, p denotes that a new number has been placed in the solution (when p reaches n^2 copies, the cycle *reset-checking* stops) and w is a witness of the application of the rule.

$$R_8 \equiv [w\, k_0 \to k_2]_e$$

$$R_9 \equiv \begin{cases} [w \to \lambda]_e \\ [k_0]_e \to \texttt{No} \end{cases}$$

If an object w has been produced, it means that at least one of the rules from the set R_7 has been applied. In other words, a new number has been placed on the solution of the sudoku and the *reset-checking* cycle must go on. In this case, objects w and k_0 are consumed by the rule from R_8 and a new object k_0 is produced. If k_0 has not been consumed by the rule from R_8, then no new number has been placed on the sudoku. This means that it does not make sense going on with the *reset-checking* cycle, since the next checking stage will have the same configuration that this one. In order to prevent an infinite sequence of cycles, object k_0 dissolves membrane e and the remaining objects w (if any) are removed.

$$R_{10} \equiv [p^{n^6}]_e \to \texttt{Yes}$$

The copies of p denote the number of squares correctly filled. When it reach n^6 copies, the membrane e is dissolved.

$$R_{11} \equiv \begin{cases} [d \to \lambda]_e \\ [k_2 \to k_1]_e \end{cases}$$

This set of rules marks the end of the checking stage. The inhibitor d is removed and the catalyst k_1 is produced so rules from R_3 can be triggered and the reset stage starts again.

Notice that membrane e is dissolved anyway. If the sudoku is not completed, but no more numbers can be placed, then the rule $[k_0]_e \to \texttt{No}$ is applied and the object `No` appears in membrane s. Otherwise, if the sudoku is completed, $[p^{n^6}]_e \to \texttt{Yes}$ is applied and `Yes` appears in membrane s. In the last step of computation, the corresponding object `Yes` or `No` is sent out to the environment.

$$R_{12}^1 \equiv \begin{cases} [\texttt{Yes}]_s \rightarrow [\,]_s\,\texttt{Yes} \\ [\texttt{No}]_s \rightarrow [\,]_s\,\texttt{No} \end{cases}$$

Also in the last step, the auxiliary objects are removed from membrane s

$$R_{12}^2 \equiv \begin{cases} [p \rightarrow \lambda]_s & [m_{ijx} \rightarrow \lambda]_s & [a_{ij} \rightarrow \lambda]_s \\ [f_{ix} \rightarrow \lambda]_s & [d \rightarrow \lambda]_s & [r_{ij} \rightarrow \lambda]_s \\ [c_{jx} \rightarrow \lambda]_s & [box_{ijk} \rightarrow \lambda]_s & [k_2 \rightarrow \lambda]_s \\ [b_{kx} \rightarrow \lambda]_s & [sq_{ij} \rightarrow \lambda]_s & \end{cases}$$

4 An Overview of the Computation

We will give some hints on the computation by following the computation of the example of order two showed in Figure 1. We start with a sudoku problem with 4 numbers placed. Such an input is encoded in the multiset $Input = z_{311}\, z_{322}\, z_{213}\, z_{141}$. The initial configuration C_0 has two membranes $[\,[\,]_e\,]_s$. Membrane s is empty and membrane e contains the input plus the objects k_1 and $f_{ix}\, c_{jx}\, b_{kx}\, sq_{ij}\, box_{ijk}, r_{ij}^4$ for $i, j, k, x \in \{1, \ldots, 4\}$.

From this configuration C_0, rules from sets R_1, R_3^r , R_5 and R_6 can be applied. Due to priority, rules from R_1 are applied and we obtain configuration C_1. The objects z_{ijx} are removed and replaced by the corresponding s_{ijx}. Four copies of object p also appear. This means that four numbers are correctly placed in the sudoku.

Rules from R_1 will nor be applied any more, since no rule produces objects z_{ijx}. From C_1 rules of set R_2 are applied and the objects $f_{31}\, c_{11}\, b_{31}\, f_{32}\, c_{22}\, b_{32}\, f_{23}\, c_{13}\, b_{13}\, f_{11}\, c_{41}\, b_{21}$ are removed from membrane e and we obtain configuration C_2.

Next, rules from R_3^r are applied (obtaining C_3) and then we apply rules from R_4 (obtaining C_4). This is the first time that we reset the system, so C_4 is identical to C_2, but with new objects d.

Since membrane e contains objects d, none of the rules from sets R_3^* nor from R_4 are applicable. The next application of the rule corresponds to R_5. The object k_1 evolves to k_0 and the checking stage begins. The new configuration C_5 contains k_0. Since no object m_{ijx} has been created yet, for each triplet (i, j, x) such that the number x can be placed in the square (i, j), one of the rules from R_6 is triggered, reaching configuration C_6. The application of one of these rules removes one copy of the corresponding r_{ij} and produces one copy of a_{ij}. Each rule only can be triggered once due to object m_{ijx}. After the application of these rules, the number of objects a_{ij} denotes the number of possible candidates to be placed in the square (i, j). In the next step, two rules from R_7 are triggered and the objects s_{124} and s_{414} are produced and configuration C_7 is reached. After the application of rules from R_6, the number of objects a_{ij} for squares $(1, 2)$ and $(4, 1)$ was exactly one. This means that for these squares, the candidate is unique. The current partial solution for the sudoku is shown in Figure 2.

4 41	42	43	44
1 31	**2** 32	33	34
3 21	22	23	24
11	**4** 12	13	**1** 14

Fig. 2. Partial solution at configuration C_7

The application of rules from R_7 has produced two copies of the object w in the configuration C_8. One of these copies, along with k_0 produces k_1 (Configuration C_9). In the next step the remaining w is deleted (Configuration C_{10}). The checking stage finishes with the application of the rule from R_{10}. Object d is removed and the configuration C_{11} is reached.

The object d is a strong inhibitor. Since it has disappeared, rules from R_3 can be applied and the reset stage starts again. We go on with this new reset stage and the application of three rules from R_7 produces the objects s_{221}, s_{423} and s_{112}. This means that three new numbers can be placed on the sudoku (see Figure 3).

4 41	**3** 42	43	44
1 31	**2** 32	33	34
3 21	**1** 22	23	24
2 11	**4** 12	13	**1** 14

Fig. 3. Partial solution at configuration C_{16}

The *checking-reset* cycle goes on and in C_{24}, the objects s_{133} and s_{442} are added. In C_{32}, s_{431}, s_{334} and s_{244} are added, and finally in C_{40}, the objects s_{232} and s_{343} are generated. Figure 4 shows the solution of the sudoku according to the objects s_{ijx} in membrane e.

The P system follows with the computation and in C_{42} there are 16 copies of objects p in membrane e. The rule from R_{10} is triggered, the membrane e is dissolved and all the objects go to membrane s. In the next step, one objects `Yes` is sent out and the remaining objects (but s_{ijx}) are deleted, so the final configuration has an object `Yes` in the environment and one membrane where the solution is encoded.

4 41	**3** 42	**1** 43	**2** 44
1 31	**2** 32	**4** 33	**3** 34
3 21	**1** 22	**2** 23	**4** 24
2 11	**4** 12	**3** 13	**1** 14

Fig. 4. Solution at configuration C_{40}

As pointed above, it is possible that for some sudokus there no exist squares with unique candidates. In such cases, when the checking stage is reached, no rule from R_7 is applied and no object w is produced. Since we do not have objects w, the rule from R_8 is not applied, and according to priority, the dissolution rule from R_9 is applied. An object `No` is sent to membrane s, the reset-checking cycle is stopped. In the next step an object `No` is sent out and the computation ends.

Notice that we have chosen an example of order 2 for illustrating the process, but the computation is similar for a sudoku problem of any order.

5 Final Remarks

P systems have showed many times to be versatile enough to represent many different situations, from real life or from more abstract scenarios. In parallel with a very expressive representation system, Membrane Computing also provides a friendly set of tools for dealing with the information. Besides the computational power of the expressiveness, the research in a new computational model also needs to face a problem of efficiency. In this paper we provide a first theoretical solution for the problem of finding a solution to a sudoku problem. It is based on an appropriate representation of the problem as a formula and a well-established solution of the SAT problem. The solution works from a theoretical point of view, but it does not make sense form a practical one.

In the second part of the paper, we have designed a solution for solving sudoku problems in an effective way. It solves all the sudokus which verifies a very common property, by following a strategy close to human-style solutions. One of the main original contributions of the design is the checking to avoid infinite loops. The implemented strategy is enough to find the solution to many sudoku problems, but it is not enough to solve all of them. The next step is to go on with the research and tackle these *hard* sudoku problems. Beyond these efforts it is the horizon of a better understanding of the cellular processes and making more and more efficient cellular designs.

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. B. Felhenhauer, F. Jarvis. Enumerating possible sudoku grids. Univ. Sheffield and Univ. Dresden, 2005. http://www.shef.ac.uk/∼pm1afj/sudoku/sudoku.pdf
2. M.A. Gutiérrez–Naranjo, M.A. Martínez–del–Amor, I. Pérez–Hurtado, M.J. Pérez–Jiménez. Solving the $N - Queens$ Puzzle with P systems. In Seventh Brainstorming Week on Membrane Computing, Vol. I, R. Gutiérrez-Escudero, M.A. Gutiérrez-Naranjo, Gh. Păun, Ignacio Pérez-Hurtado, A. Riscos Núñez, (eds.) Fénix Editora (2009) 199-210.
3. I. Lynce, J.Ouaknine. Sudoku as a SAT Problem. Proceedings of the 9 th International Symposium on Artificial Intelligence and Mathematics, AIMATH 2006, Fort Lauderdale (2006).
   http://anytime.cs.umass.edu/aimath06/proceedings/P34.pdf
4. 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) 284-294.
5. T. Yato and T. Seta. Complexity and Completeness of Finding Another Solution and Its Application to Puzzles. IEICE Trans. Fundamentals, E86-A (5) (2003) 1052–1060.
6. http://www.dellmagazines.com
7. http://www.nikoli.co.jp/puzzles/