Solving SAT in Linear Time with a Neural-like Membrane System

Juan Pazos, Alfonso Rodríguez-Patón , and Andrés Silva

Facultad de Informática, Campus de Montegancedo s/n,
Boadilla del Monte - 28660 Madrid (SPAIN).
arpaton@fi.upm.es

Abstract. We present in this paper a neural-like membrane system solving the SAT problem in linear time. These neural P systems are nets of cells working with multisets. Each cell has a finite state memory, processes multisets of symbol-impulses, and can send impulses ("excitations") to the neighboring cells. The maximal mode of rules application and the replicative mode of communication between cells are at the core of the efficiency of these systems.

1 Introduction

The present paper deals with a topic for further research (open problem) addressed in the paper [11]. In that paper a parallel and distributed computational model called *tissue P System* (in short, *tP system*) was introduced and defined. The efficient resolution of NP-complete problems over graphs was proposed like one possible application of these new systems. For example, it was proved that the Hamiltonian Path Problem can be solved in linear time with these tP systems (details in [11]). However, it was also proposed in that paper like a "topic for further research" the search for any other type of problems outside graph theory that could be efficiently solved by the tP systems.

In this paper we show that tP systems (also called *neural P systems* or *nP systems* in [15]) can solved in linear time a very general and classical NP-complete problem: the SAT problem.

tP systems can be seen at the same time as a contribution to neural networks (of a symbolic type), to membrane computing (with cells arranged in "tissues"), to finite automata networks (working not with strings, but with multisets of symbols), to multiset processing, to (distributed) automata and language theory. The motivation is two-fold: the inter-cellular communication (of chemicals, energy, information) by means of complex networks of protein channels (see, e.g., [1], [10]), and the way the neurons co-operate, processing impulses in the complex net established by synapses (see, e.g., [1], [2]).

The common mathematical model of these two kinds of symbol-processing mechanisms is a net of finite state devices: networks of finite-automata-like processors, dealing with symbols, according to local states (available in a finite

number for each "cell"), communicating through these symbols, along channels ("axons") specified in advance. Note that the neuron modelling was the starting point of the theory of finite automata ([13], [8]), that symbol processing neural networks have a rich (and controversial) history (see [4] and its references), and that networks of string-processing finite automata have appeared in many contexts ([5], [7], [12], etc), but our models are different in many respects from all these previous models.

Having in mind the bio-chemical reality we refer to, a basic problem concerns the organization of the bunch of symbols available in each node, and the easiest and most natural answer is: no organization. Formally, this means that we have to consider *multisets* of symbols, sets with multiplicities associated with their elements. In this way, we need a kind of finite automata dealing with multisets of symbols, a topic which falls into an area of (theoretical) computer science not very much developed, although some recent (see, e.g., [6]), or not so recent (see, e.g., [3]) approaches can be found in the literature. Actually, most of the vivid area of membrane computing (P systems) [14, 15] is devoted to multiset processing (details at `http://psystems.disco.unimib.it/`).

The computing models proposed in [11], under the name of *tP systems*, consist of several *cells*, related by protein channels. In order to preserve also the neural intuition, we will use the suggestive name of *synapses* for these channels. Each cell has a state from a given finite set and can process multisets of *objects*, represented by symbols from a given alphabet. The standard rules are of the form $sM \to s'M'$, where s, s' are states and M, M' are multisets of symbols. Some of the elements of M' may be marked with the indication "go", and this means that they have to immediately leave the cell and pass to the cells to which we have direct links through synapses. This communication (transfer of symbol-objects) can be done in a replicative manner (the same symbol is sent to all adjacent cells), or in a non-replicative manner; in the second case we can send all the symbols to only one adjacent cell, or we can distribute them, nondeterministically. One more choice appears in using the rules $sM \to s'M'$: we can apply such a rule only to one occurrence of M (that is, in a sequential, *minimal* way), or to all possible occurrences of M (a *parallel* way), or, moreover, we can apply a maximal package of rules of the form $sM_i \to s'M_i', 1 \leq i \leq k$, that is, involving the same states s, s', which can be applied to the current multiset (the *maximal* mode). By the combination of the three modes of processing objects and the three modes of communication among cells, we get nine possible behaviors of our machinery.

A way to use such a computing device is to start from a given initial configuration (that is, initial states of cells and initial multisets of symbol-objects placed in them) and to let the system proceed until reaching a halting configuration, where no further rule can be applied, and to associate a result with this configuration. Because of the nondeterminism, starting from one given initial configuration we can reach arbitrarily many different halting configurations, hence we can get arbitrarily many outputs. Another possibility is to also provide *inputs*, at various times of a computation, and to look for the outputs related to

them. Here we will consider only the first possibility, of *generative* tP systems, and the output will be defined by sending symbols out of the system.

At the first sight, such a machinery (a finite net of finite state devices) seems not to be very powerful, e.g., as compared with Turing machines. Thus, it is rather surprising to find that tP systems with a small number of cells (two or four), each of them using a small number of states (resp., at most five or four) can simulate any Turing machine, even in the non-cooperative case, that is, only using rules of the form $sM \to s'M'$ with M being a singleton multiset; moreover, this is true for all modes of communication for the minimal mode of using the rules, and, in the cooperative case, also when using the parallel or the maximal mode of processing objects. When the rules are non-cooperative and we use them in the maximal mode, a characterization of Parikh images of ET0L languages is obtained, which completes the study of the computing power of our devices (showing that in the *parallel* and *maximal* cases we dot not get computational universality).

The above mentioned results obtained in [11] indicate that our cells are "very powerful"; as their power lies in using states, hence in remembering their previous work, a natural idea is to consider tP systems with a low bound on the number of states in each cell. In view of the previously mentioned results, tP systems with at most 1, 2, 3, or 4 states per cell are of interest. We only briefly consider this question here, and we show that even reduced tP systems as those which use only one state in each cell can be useful: using such a net we can solve the Satisfiability Problem in linear time (this is a direct consequence of the structure of a tP system, of the maximal mode of processing objects, and of the power of replicating the objects sent to all adjacent cells); remember that SAT is an NP-complete problem.

The power of tP systems with a reduced number of states per component remains to be further investigated. Actually, many other natural research topics can be considered, with motivations from automata and language theory (variants, power, normal forms), neural networks (learning, dynamic sets of neurons, dynamic synapses), computability (other NP-complete problems treated in this framework), dynamic systems (reachable configurations), etc.

2 Tissue P Systems

We now pass to the definition of our variant of membrane (P) systems, which can also be considered as a model of a symbolic neural net. We introduce it in the general form, then we will consider variants of a restricted type.

A *tissue P system* or a *neural P system* depending the motivation, in short, a tP or nP system, of *degree* $m \geq 1$, is a construct

$$\Pi = (E, \sigma_1, \ldots, \sigma_m, syn, i_{out}), \text{ where}$$

1. E is a finite non-empty alphabet (of *chemical objects*, but we also call them *excitations/impulses*);

2. $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ (*synapses* among cells);
3. $i_{out} \in \{1, 2, \ldots, m\}$ indicates the *output cell*;
4. $\sigma_1, \ldots, \sigma_m$ are *cells*, of the form $\sigma_i = (Q_i, s_{i,0}, w_{i,0}, P_i), 1 \leq i \leq m$, where:
   (a) Q_i is a finite set (of *states*);
   (b) $s_{i,0} \in Q_i$ is the *initial state*;
   (c) $w_{i,0} \in E^*$ is the *initial multiset* of impulses;
   (d) P_i is a finite set of *rules* of the form $sw \to s'xy_{go}z_{out}$, where $s, s' \in Q_i$, $w, x \in E^*$, $y_{go} \in (E \times \{go\})^*$ and $z_{out} \in (E \times \{out\})^*$, with the restriction that $z_{out} = \lambda$ for all $i \in \{1, 2, \ldots, m\}$ different from i_{out}.

A tP system as above is said to be *cooperative* if it contains at least a rule $sw \to s'w'$ such that $|w| > 1$, and *non-cooperative* in the opposite case.

Any m-tuple of the form $(s_1w_1, \ldots, s_mw_m)$, with $s_i \in Q_i$ and $w_i \in E^*$, for all $1 \leq i \leq m$, is called a *configuration* of Π; $(s_{1,0}w_{1,0}, \ldots, s_{m,0}w_{m,0})$ is the *initial configuration* of Π.

Using the rules from the sets $P_i, 1 \leq i \leq m$, we can define *transitions* among configurations. To this aim, we first consider three *modes of processing the stimuli* and three *modes of transmitting excitations* from a cell to another one. Let us denote $E_{go} = \{(a, go) \mid a \in E\}$, $E_{out} = \{(a, out) \mid a \in E\}$, and $E_{tot} = E \cup E_{go} \cup E_{out}$. For $s, s' \in Q_i, x \in E^*, y \in E^*_{tot}$, we write

$$
\begin{aligned}
sx \Longrightarrow_{min} s'y \text{ iff } & sw \to s'w' \in P_i, w \subseteq x, \text{ and } y = (x - w) \cup w', \\
sx \Longrightarrow_{par} s'y \text{ iff } & sw \to s'w' \in P_i, w^k \subseteq x, w^{k+1} \not\subseteq x, \\
& \text{for some } k \geq 1, \text{ and } y = (x - w^k) \cup w'^k, \\
sx \Longrightarrow_{max} s'y \text{ iff } & sw_1 \to s'w'_1, \ldots, sw_k \to s'w'_k \in P_i, k \geq 1, \\
& \text{such that } w_1 \ldots w_k \subseteq x, y = (x - w_1 \ldots w_k) \cup w'_1 \ldots w'_k, \\
& \text{and there is no } sw \to s'w' \in P_i \text{ such that } w_1 \ldots w_k w \subseteq x.
\end{aligned}
$$

In the first case, only one occurrence of the multiset from the left hand side of a rule is processed (replaced by the multiset from the right hand of the rule, at the same time changing the state of the cell), in the second case a maximal change is performed with respect to a chosen rule, in the sense that as many as possible copies of the multiset from the left hand side of the rule are replaced by the corresponding number of copies of the multiset from the right hand side, while in the third case a maximal change is performed with respect to all rules which use the current state of the cell and introduce the same new state after processing the impulses.

We also write $sx \to_\alpha sx$, for $s \in Q_i, x \in E^*$, and $\alpha \in \{min, par, max\}$, if there is no rule $sw \to s'w'$ in P_i such that $w \subseteq x$. This encodes the case when a cell cannot process the current impulses in a given state (it can be "unblocked" after receiving new impulses from its ancestors).

The multiset w' from a rule $sw \to s'w'$ contains symbols from E, but also symbols of the form (a, go) (or, in the case of cell i_{out}, of the form (a, out)). Such symbols will be sent to the cells related by synapses to cell σ_i where the rule $sw \to s'w'$ is applied, according to the following modes:

- *replicative* (indicated by *repl*): each symbol a, for (a, go) appearing in w', is sent to each of the cells σ_j such that $(i, j) \in syn$;
- *unique destination* (indicated by *one*): all symbols a appearing in w' in the form (a, go) are sent to one of the cells σ_j such that $(i, j) \in syn$, nondeterministically chosen; more exactly, in the case of modes *par* and *max* of using the rules, we first perform all applications of rules, and after that we send all obtained symbols to a unique descendant of the cell (that is, we do not treat separately the impulses introduced by each rule, but all of them in a package);
- *non deterministic distribution* (indicated by *spread*): the symbols a appearing in w' in the form (a, go) are non-deterministically distributed among the cells σ_j such that $(i, j) \in syn$.

In order to formally define the transition among the configurations of Π we need some further notations. For a multiset w over E_{tot}, we denote by $go(w)$ the multiset of symbols $a \in E$ appearing in w in the form (a, go), and by $out(w)$ the multiset of symbols $a \in E$, appearing in w in the form (a, out). Clearly, $go(w)(a) = w((a, go))$ and $out(w)(a) = w((a, out)), a \in E$. Moreover, for a node i in the graph defined by syn we denote $ant(i) = \{j \mid (j, i) \in syn\}$ and $succ(i) = \{j \mid (i, j) \in syn\}$ (the ancestors and the successors of node i, respectively).

During any transition, some cells can do nothing: if no rule is applicable to the available multiset of impulses in the current state, then a cell waits until new impulses are sent to it from its ancestor cells.

A sequence of transitions among configurations of the tP system Π is called a *computation* of Π. A computation which ends in a configuration where no rule in no cell can be used, is called a *halting* computation. Assume that during a halting computation the tP system Π sends out, through the cell $\sigma_{i_{out}}$, the multiset z. We say that the vector $\Psi_E(z)$, representing the multiplicities of impulses from z, is *computed* (or *generated*) by Π.

Rather surprising, if we take into consideration the apparently weak ingredients of our models, when using the mode *min* of applying the rules, even the non-cooperative tP systems turn out to be computationally universal. More results about the computational power of the differents variants of tP systems are in [11] and [15].

3 Solving SAT in Linear Time

Problems related to paths in a (directed) graph can be easily solved by a tP system, just by constructing a net with the synapses graph identical to the graph we deal with, constructing all paths in the graph with certain properties by making use of the maximal mode of applicating the rules and of the replicative communication, and checking the existence of a path with a desired property. The HPP is solved in this way in [11].

The architecture of tP systems and their way of working (especially the fact that in the maximal mode of using the rules we can process all impulses which

may be processed in such a way that the same next state is obtained, irrespective which rules are used, and the fact that in the replicative mode one can send the same impulses to all successors of a cell) have an intrinsic computational power. We will show this power with the resolution of the SAT problem.

Let $F = C_1 \wedge C_2 \wedge \ldots \wedge C_m$, where each clause $C_i, 1 \leq i \leq m$, is a disjunction $C_i = y_1 \vee y_2 \vee \ldots \vee y_r$, with each y_j being either a propositional variable, x_s, or its negation, $\neg x_s$, for $s \in \{1, 2, \ldots, n\}$. The SAT problem ask wether or not there is a truth-assignment of the variables that makes the formula true. Note: We will use t_i (respectively f_i) to represent $x_i = true$ (resp. $x_i = false$).

We construct the tP system Π with the following components:

$$\Pi = (E, \sigma_S, \sigma_{t_1}, \sigma_{f_1}, \ldots, \sigma_{t_n}, \sigma_{f_n}, \sigma_E, \sigma_{y_{ji}}, \sigma_O, syn, O),$$

where $y_{j,i}$ are the variables y_j in the clause C_i for $j \in \{1, 2, \ldots, n\}$ and $i \in \{1, 2, \ldots, m\}$.

$$\begin{aligned}
E &= \{z \mid z = \{\lambda\} \text{ or } z = g_1 g_2 \cdots g_i, \text{ for } g_i = t_i, f_i, 1 \leq i \leq n\},\\
\sigma_S &= (\{s\}, s, \lambda, \{s\lambda \rightarrow s(\lambda, go)\})\\
\sigma_{t_i} &= (\{s\}, s, t_i, \{sz \rightarrow s(zt_i, go), \text{ for each } i = \{1, 2, \ldots, n\}, \text{ and}\\
&\qquad z = \lambda \text{ or } z = g_1 g_2 \cdots g_i, \text{ for } g_i = t_i, f_i, 1 \leq i \leq n\}),\\
\sigma_{f_i} &= (\{s\}, s, f_i, \{sz \rightarrow s(zf_i, go), \text{ for each } i = \{1, 2, \ldots, n\}, \text{ and}\\
&\qquad z = \lambda \text{ or } z = g_1 g_2 \cdots g_i, \text{ for } g_i = t_i, f_i, 1 \leq i \leq n\}),\\
\sigma_E &= (\{s\}, s, \lambda, \{sz \rightarrow s(z, go)\} \mid z = g_1 g_2 \cdots g_n, |z| = n),\\
\sigma_{y_{j,i}} &= (\{s\}, s, \lambda, \{sz \rightarrow s(z, go) \text{ if } t_j \in z \text{ and } C_i \text{ contains } x_j,\\
&\qquad \text{or if } f_j \in z \text{ and } C_i \text{ contains } \neg x_j\}),\\
\sigma_O &= (\{s\}, s, \lambda, \{sz \rightarrow s(z, out)\}),\\
syn &= \{(S, g_1), (g_k, g_{k+1}), (g_n, E), (E, y_{j,1}), (y_{j,i}, y_{j,i+1}), (y_{j,m}, O) \mid\\
&\quad g_k = t_k, f_k \text{ for } k = \{2, \ldots, n-1\}, i = \{2, \ldots, m-1\} \text{ and } j = \{1, 2, \ldots, n\}\},\\
O &= \text{ Output membrane.}
\end{aligned}$$

It is easy to see that $N_{max,repl}(\Pi) \neq \emptyset$ if and only if the SAT problem has a solution. This system works as follows. There are two phases. In the first one the system generates all the truth-assignments in the form of strings of length n composed of $t_i, f_i, 1 \leq i \leq n$ in all possible combinations (this takes n steps). The strings are constructed starting in membrane S with sequential concatenatations. In each membrane g_i the corresponding symbol g_i for $g_i = t_i, f_i, 1 \leq i \leq n$ is concatenated and the resulting strings are replicated to the following membranes g_{i+1}. The generation phase ends with the addition of the last symbol $g_n = t_n, f_n$ forming the 2^n truth-assignments z that reach the membrane E. Note that the strings z are manipulated as symbols through the tP system. In figure 1 we can see the topology of the generative membranes.

In membrane E starts the second phase: the computational or "filtering" phase (resembling the filter steps of the Lipton's algorithm, see [9]). This second phase is a sequence of m steps or filters, one for each clause of the formula. In

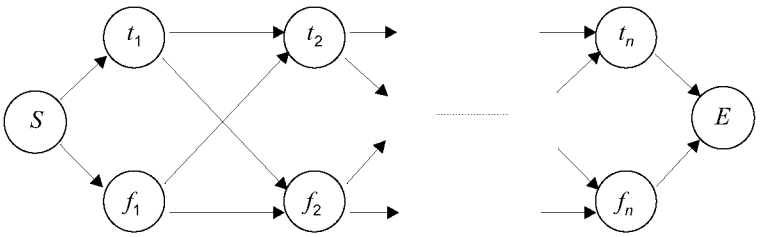

Fig. 1. Generation of the the 2^n truth-assignments z of length n.

the first step (membranes $y_{j,1}$) only the strings z that satisfies the first clause C_1 are allow to go to the next level. In other words, only the strings z such that $t_j \in z$ and C_1 contains x_j, or the strings z such that $f_j \in z$ and C_i contains $\neg x_j$ are allow to travel to the second level (filter). These filters are repeated until the level m. If some z reaches the output membrane O, the formula is satisfiable and the symbol z sent out of the system encodes a SAT assignment. If no symbol z is sent out after $n+m$ steps, the formula is no satisfiable. For example, the filter membranes for the formula $F = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2)$ are shown in figure 2.

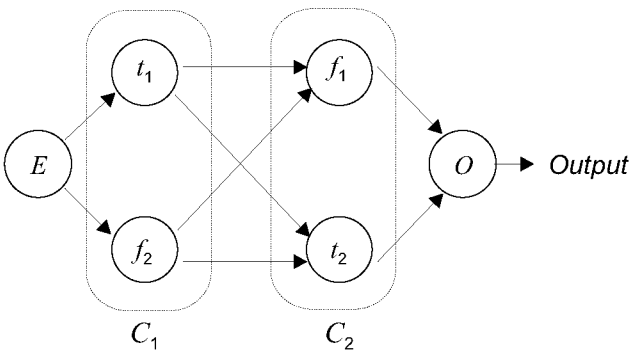

Fig. 2. Computation phase: two "filtering" steps across membranes.

Therefore, our tP system (or nP system) can solve the SAT problem in linear time: n steps for generating all truths-assignments and m steps to check the truth value of each of the m clauses. (Note that in our calculations of time steps we have not count the two steps to/from cell E. We could suppress the E cell but for sake of clarity we have maintained it in the system acting like delimiter between the two phases).

4 Conclusions

We have shown that neural-like membrane systems can solve in linear time not only graph theory problems but also the SAT problem. There are a lot of topics for further research pointed in [11]. One more topic for further research could be the interpretation of the SAT bio-algorithm presented here like a special ballistic computation. In some sense the objects z cross the obstacles imposed by the clauses. The SAT problem has solution if some z arrives at the end of the "clause labyrinth".

References

1. B. Alberts et al., *Essential Cell Biology. An Introduction to the Molecular Biology of the Cell*, Garland Publ. Inc., New York, London, 1998.
2. M.A. Arbib, *Brains, Machines, and Mathematics*, second ed., Springer-Verlag, Berlin, 1987.
3. J.P. Banatre, P. Fradet, D. LeMetayer, Gamma and the chemical abstract reaction model: fifteen years after, in vol. *Multiset Processing. Mathematical, Computer Science, and Molecular Computing Points of View* (C.S. Calude, Gh. Păun, G. Rozenberg, A. Salomaa, eds.), *Lecture Notes in Computer Science*, 2235, Springer-Verlag, 2001, 17–44.
4. D.S. Blank *et al* (24 co-authors), Connectionist symbol processing: Dead or alive?, *Neural Computing Surveys*, 2 (1999), 1–40.
5. C. Choffrut, ed., *Automata Networks*, *Lecture Notes in Computer Science*, 316, Springer-Verlag, Berlin, 1988.
6. E. Csuhaj-Varju, C. Martin-Vide, V. Mitrana, Multiset automata, in vol. *Multiset Processing. Mathematical, Computer Science, and Molecular Computing Points of View* (C.S. Calude, Gh. Păun, G. Rozenberg, A. Salomaa, eds), *Lecture Notes in Computer Science*, 2235, Springer-Verlag, 2001, 67–82.
7. F. Gecseg, *Products of Automata*, Springer-Verlag, Berlin, 1986.
8. S.C. Kleene, Representation of events in nerve nets and finite automata, *Automata Studies*, Princeton Univ. Press, Princeton, N.J., 1956, 2–42.
9. R. J. Lipton, Using DNA to solve NP-complete problems, *Science*, 268 (April 1995), 542–545.
10. W.R. Loewenstein, *The Touchstone of Life. Molecular Information, Cell Communication, and the Foundations of Life*, Oxford Univ. Press, New York, Oxford, 1999.
11. Carlos Martín-Vide, Gheorghe Păun, Juan Pazos and Alfonso Rodríguez-Patón, Tissue P systems, *Theoretical Computer Science*, vol. 296, issue 2, (2003), 295–326.
12. A. Mateescu, V. Mitrana, Parallel finite automata systems communicating by states, *Intern. J. Found. Computer Sci.*, to appear.
13. W.S. McCulloch, W.H. Pitts, A logical calculus of the ideas immanent in nervous activity, *Bull. Math. Biophys.*, 5 (1943), 115–133.
14. Gh. Păun, Computing with membranes, *Journal of Computer and System Sciences*, 61, 1 (2000), 108–143 (see also *Turku Center for Computer Science-TUCS Report* No 208, 1998, www.tucs.fi).
15. Gh. Păun, *Membrane Computing*. Springer-Verlag, 2002.