International Journal of Computer Applications (0975 – 8887)
Volume 54– No.13, September 2012

A Cloud based P Systems Algorithm

Eamd Nabil
Computer Science Dept.
MUST University
Cairo- Egypt

Hala Hameed
Computer Science Dept.
MUST University
Cairo- Egypt

Amr Badr
Computer Science Dept.
Cairo University
Cairo-Egypt

ABSTRACT
A P system is a computability model which is biochemically inspired, it is a general distributed model, highly parallel, nondeterministic, based on the notion of a membrane structure. Till this moment, there is no exact idea about the real implementation of P systems. P systems are used in solving NP-complete problems in polynomial time, but with building the whole exponential search space. Cloud computing assume infinite memory and infinite processing power. This paper proposes an algorithm that uses the cloud resources in a fully parallel manner as a step towards P systems implementation, the nondeterminism property of P systems is certainly not maintained. The paper used the SAT problem as the case study.

General Terms
Membrane computing, Artificial Intelligence, Algorithms, Optimization, Satisfiability problem, Natural Computing, Cloud computing.

Keywords
P systems, Membrane computing, Cloud computing, SAT problem.

1. INTRODUCTION
Membrane computing is the branch of natural computing which investigates computing models that are abstracted from the structure and functioning of living cells as well as from their interactions in tissues or higher-order biological structures [1].

P system structure consists of several cell-like membranes; these membranes are placed inside a unique "skin" membrane. The structure is represented using Venn diagram without intersected sets and with a unique superset. The structure has multiple regions delimited by the membranes. In each region, there are some objects which could evolve and transform into other objects. Very important property of a P system is that it works in a distributed, maximally parallel, selective commutative and nondeterministic fashion [2].

Although P systems achieved a great success in the theoretical design of solutions to NP-complete problems, these solutions have a fundamental drawback from a practical point of view. It is not clear yet what is the actual real implementation of Membrane Computing. It could be implemented *in vitro, in vivo* or *in silico.* In any case, a membrane will have a space associated that could be a piece of memory in a computer, a pipe in a lab, or a volume of bacteria. Only brute force algorithms will be able to implement little instances of such problems. If we take an *in vivo* implementation where each feasible solution will be encoded in an elementary membrane, and such elementary membrane is implemented in a bacteria of mass $\cong 7\times10^{-16}$ kg., for example ***E. Coli***. Then, a brute force algorithm which solves an instance of an NP problem with input size 40, for example, it will need approximately a mass $\cong 6 \times 10^{24}$ kg., which equals approximately mass of the earth. So, it is not practical for a P system to be implemented in solving relatively large NP-Complete problems.

Despite the difficultiesthat are related to P systems nondeterminism, attempts have been done to implement P systems on dedicated reconfigurable hardware, as done by Petreska & Teuscher [3], or on a cluster of computers, as performed by Ciobanu& Guo [4], or in a distributed fashion, as reported by Syropoulos *et al.* [5]. All these studies made use of P system model,yet, our study can be considered a better approximation for the P system model implementation, as we merge the cloud space into our work.

According to the aforementioned problem of real implementation of P systems -till now, we proposed applying P systems on the cloud computing model as we mentioned before. Cloud computing model assumes –theoretically- that infinite resources Cloud based implementation of P systems maintains the maximal parallel processing property, but of course, nondeterminism is not maintained because the cloud model consists of a huge amount of deterministic Turing machines.

In this paper, an algorithm is proposed to solve satisfiability (SAT) problem which is one of the most common NP-complete problems. The proposed algorithm solves SAT problem as a case study, but of course it could be generalized over other NP-complete.

The paper is organized as follows: Section 2 gives an introduction to P systems, while section 3 explains the cloud computing model. In section 4, the SAT problem is illustrated by an example. Section 5 depicts the proposed cloud based P systems algorithm. Finally section 6 introduces conclusions and future work.

2. INTRODUCTION TO P SYSTEMS
Membrane computing has been inspired by the structure of a simple living cell. Cell means membranes. The contents of a living cell are separated from its surrounding by an external membrane. Inside the living cell itself, there are several compartments where independent (and distributed) biochemical processes can take place in maximally parallel manner. Communication is possible, as objects can transfer from time to time and from one compartment to another inside the cell through the internal membranes. Objects can also transfer from the cell inner side to the outside of the cell and vice versa, through the external membrane [2, 6, 7].

Non-determinism occurs as a result of the unknown sequence of objects processing order (undergoing biochemical reactions) which can differ from time to time. Moreover, living cells have put back the focus onto the notion of multi-setobjects where a cell contains substances swim in an aqueous solution. In such circumstances, the concentration of an object is important, in computer science, the concentration of an object has been translated to multiplicities of an object, hence, multi-sets. This model of a living cell has been the inspiration for computing devices that can work in distributed, maximally parallel Nondeterministic manner [2, 8, 9].

Processing is done by firing rules rather than executing commands/statements. Firing of rules takes the form of running biochemical reactions between reactors. A reaction/rule cannot be fired unless there is sufficient amount of the reactors existing in the current compartment.

A priority relation between evolution rules can be considered [1]. The evolution is done in parallel for all objects that are capable of evolving. This way, we can obtain a computing device that starts with a certain number of objects and then it is continuously increased through the evolution process. If the system halts (i.e. no objects can further evolve), then the computation is finished, with the result given as the number of objects in a specified membrane. If the development of the system goes forever, then the computation fails to have an output. Also membrane division can be considered. This way, an enhanced parallelism is obtained, which can be very useful from the computational complexity point of view. A membrane can dissolve and its contents are sent to the parent membrane [1].

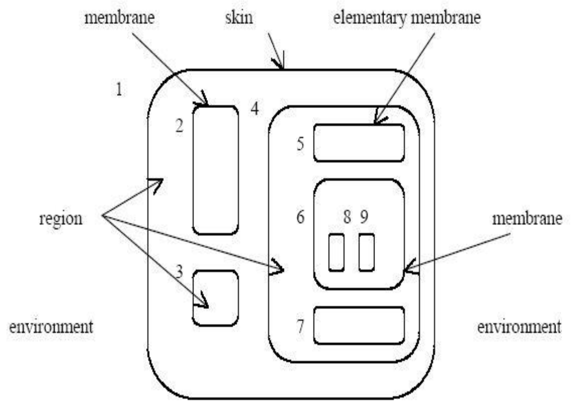

Fig 1: Basic Membrane structure

There are many variants of P systems, below is the class of P systems with active membranes [1], which could be constructed in the following form:

$$\Pi = (O, H, \mu, w_1, \dots, w_m, R)$$

Where:

1. $m \leftarrow 1$ (the initial degree of the system);
2. O is the alphabet of *objects*;
3. H is a finite set of *labels* for membranes;
4. μ is a *membrane structure*, consisting of m membranes initially having neutral polarizations, labeled with elements of H;
5. $w_1, \dots\dots, w_m$ are strings over O, describing the *multisets of objects* placed in the m regions of μ;
6. R is a finite set of *developmental rules*, of the following forms:

a) $[a \rightarrow v]_h^e$, for $h \in H, e \in \{+,-,0\}, a \in O, v \in O^*$
Object evolution rules, associated with membranes and depending on the label and the charge of the membranes. Hint: For simplicity, the label is written only onceoutside the brackets and the internal label is omitted.

b) (b) $a[\ ]_h^{e1} \longrightarrow [b]_h^{e2}$, for $h \in H, e \in \{+,-,0\}, a, b \in O$
Communication rules: An object is introduced in the membrane, and possibly modified during this process; also the polarization of the membrane can be modified, but not its label.

c) $[a]_h^{e1} \longrightarrow [\ ]_h^{e2} b$, for $h \in H, e_1, e_2 \in \{+,-,0\}, a, b \in O$
Out communication rules: An object is sent out of the membrane, and possibly modified during this process; also the polarization of the membrane can be modified, but not its label.

d) $[a]_h^e \longrightarrow b$, for $h \in H, e \in \{+,-,0\}, a, b \in O$
Dissolving rules: In reaction with an object, a membrane can be dissolved, while the object specified in the rule can be modified.

e) $[a]_h^{e1} \longrightarrow [b]_h^{e2}[c]_h^{e3}$, for $h \in H, e_1, e_2, e_3 \in \{+,-,0\}, a, b, c \in O$
Division rules for elementary membranes: In reaction with an object, the membrane is divided into two membranes with the same label, and possibly of different polarizations; the object specified in the rule is replaced in the two new membranes possibly by new objects; the remaining objects are duplicated and may evolve in the same step by rules of type (a).It is possible to allow the change of membrane labels. For instance, a division rule can be of the below more general form.

$$[a]_{h1}^{e1} \longrightarrow [b]_{h2}^{e2}[c]_{h3}^{e3}, \text{for } h_1, h_2, h_3 \in H, e_1, e_2, e_3 \in \{+,-,0\}, a, b, c \in O$$

3. CLOUD COMPUTING

Cloud computing is the delivery of computing and storage capacity as a service [10] to a community of end-recipients. The name comes from the use of a cloud-shaped symbol as an abstraction for the complex infrastructure it contains in system diagrams. Cloud computing entrusts services with a user's data, software and computation over a network [11].

There are three types of cloud computing model [11]:

1. **Infrastructure as a Service (IaaS)**
Infrastructure as a Service is a provision model in which an organization outsources the equipment used to support operations including storage, hardware, servers and networking components. The service provider owns the equipment and is responsible for its housing, running and maintaining. The client typically pays on a per-use basis.

2. **Platform as a Service (PaaS)**
Platform as a Service (PaaS) is a way to rent hardware, operating systems, storage and network capacity over the Internet. The service delivery model allows the customer to rent virtualized servers and associated services for running existing applications or developing and testing new ones.

3. **Software as a Service (SaaS)**
   Software as a Service (SaaS) is a software distribution model in which applications are hosted by a vendor or service provider and made available to customers over a network, typically the internet.

The proposed algorithm makes use of cloud as (platform as a service). The algorithm could be written, for example, in c# programming language and Windows azure platform.

4. THE SATISFIABILITY PROBLEM

Satisfiability (SAT) problem is one of the Constraint satisfaction problems (CSPs), and it is considered an NP-complete problem. It consists of a logical propositional formula in n variables and the requirement is to assign a value (true or false) for each variable that makes the formula true [12, 13]. Equally important is to determine whether no such assignments exist, which would imply that the function expressed by the formula is identically false for all possible variable assignments. In this latter case, we would say that the function is unsatisfiable, otherwise it is satisfiable.

In complexity theory, the satisfiability problem (SAT) is a decision problem, whose instance is a Boolean expression written using only AND, OR, NOT, variables, and parentheses [12]. A wide range of other naturally occurring decision and optimization problems can be transformed into instances of SAT [14]. A class of algorithms called SAT solvers can efficiently solve a large enough subset of SAT instances to be useful in various practical areas such as circuit design and automatic theorem proving, by solving SAT instances made by transforming problems that arise in those areas.

For k-SAT, the formula consists of a conjunction of clauses and each clause is a disjunction of k variables, any of which may be negated.

3-satisfiability (3-SAT) is a special case of*k*-satisfiability ***(k-SAT)*** or simply satisfiability (SAT), when each clause contains exactly $k = 3$ literals. A literal is either a variable or the negation of a variable (the negation of an expression can be reduced to negated variables by De Morgan's laws. For example, $\boldsymbol{x_1}$ is a positive literal and *not(X₂)* is a negative literal. A clause is a disjunction of literals, e.g. $x_1 \lor not(x_2)$.

Here is an example, where ¬ indicates negation.

$$E = (x_1 \lor \neg x_2 \lor \neg x_3) \land (x_1 \lor x_2 \lor x_4)$$

E has two clauses (denoted by parentheses), four variables (x_1, x_2, x_3, x_4), and k=3 (three literals per clause). To solve this instance of the decision problem we must determine whether there is a truth value (true or false) that we can assign to each of the variables (x_1 through x_4) such that the entire expression is true. In this instance, there is such an assignment (x_1 = true, x_2 = true, x_3=true, x_4=true), so the answer to this instance is yes. This is one of many possible assignments, for instance, any set of assignments including $x_1 = true$ being sufficient. If there were no such assignment(s), the answer would be no.

5. THE PROPOSED ALGORITHM

The proposed algorithm treats each instance on the cloud as a membrane; membrane division is implemented by sending two messages to two other new instances from the pool of instances on the cloud. The sender instance (virtual membrane) is automatically deallocated (dissolved). P systems have the ability to solve NP-complete problems by creating and working on exponential search space in polynomial or linear time. The same idea is implemented here where we substitute membranes by virtual ones, namely cloud instances. From P systems, all development rules exist in membranes are executed in parallel, thus parallelism is totally maintained as all instances are working independently on the same time.

The cloud based P system algorithm makes use of the two rules (a) and (e) explained in section 2. We also add a new rule implemented in the proposed algorithm where a membrane with its contents is vanished. The vanishing rule could be expressed in P systems language as follows:

$$[\,a\,]_h^e \longrightarrow \lambda, \quad \text{for } h \in H, e \in \{+,-,0\}, a, b \in O$$

A flow chart of the cloud based P system algorithm is depicted in figure 2.

Here is a simple example that illustrates how the algorithm is working. Consider the following 3-SAT problem:

$$(x_1 \lor x_2 \lor \neg x_3) \land (x_5 \lor x_6 \lor \neg x_7) \land (\neg x_4 \lor \neg x_8 \lor \neg x_9)$$

The structure of the solution can be represented as an array, in our example the array is of length 9 binary digits as depicted in figure 3, each digit can have two possible values, namely 0 or 1, for false and true respectively.

The root instance contains an algorithm and a data structure. The solution array started empty and is initialized gradually. The exponential search space is created in linear time. Each solution resides in an independent node (instance) together with an algorithm that modifies the solution and checks if the solution is valid or not (figure 2).

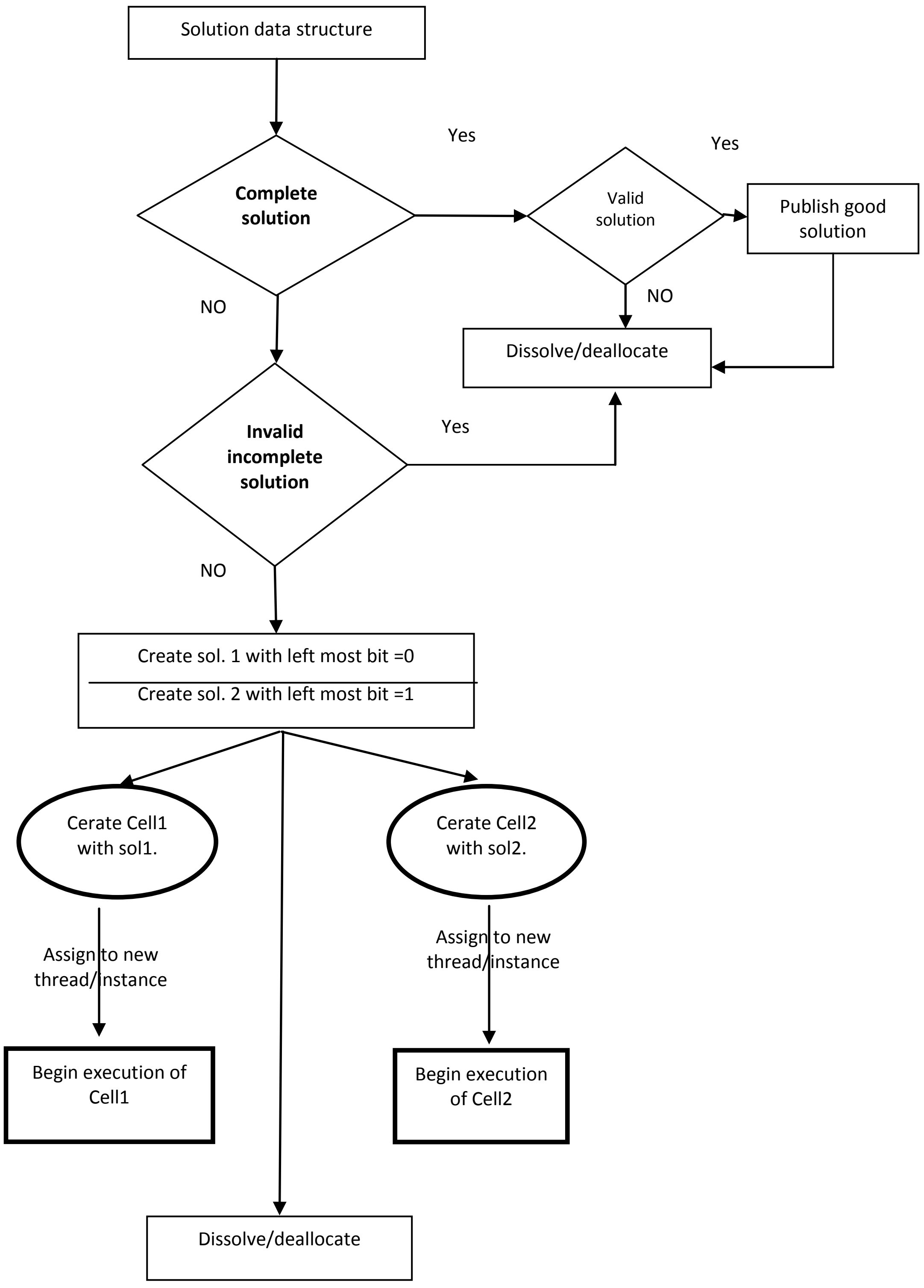

Fig 2: An illustration of the cloud based P system algorithm

0	0	1	1	0	0	1	0	1

Fig 3: An example of a solution consists of 9 digits

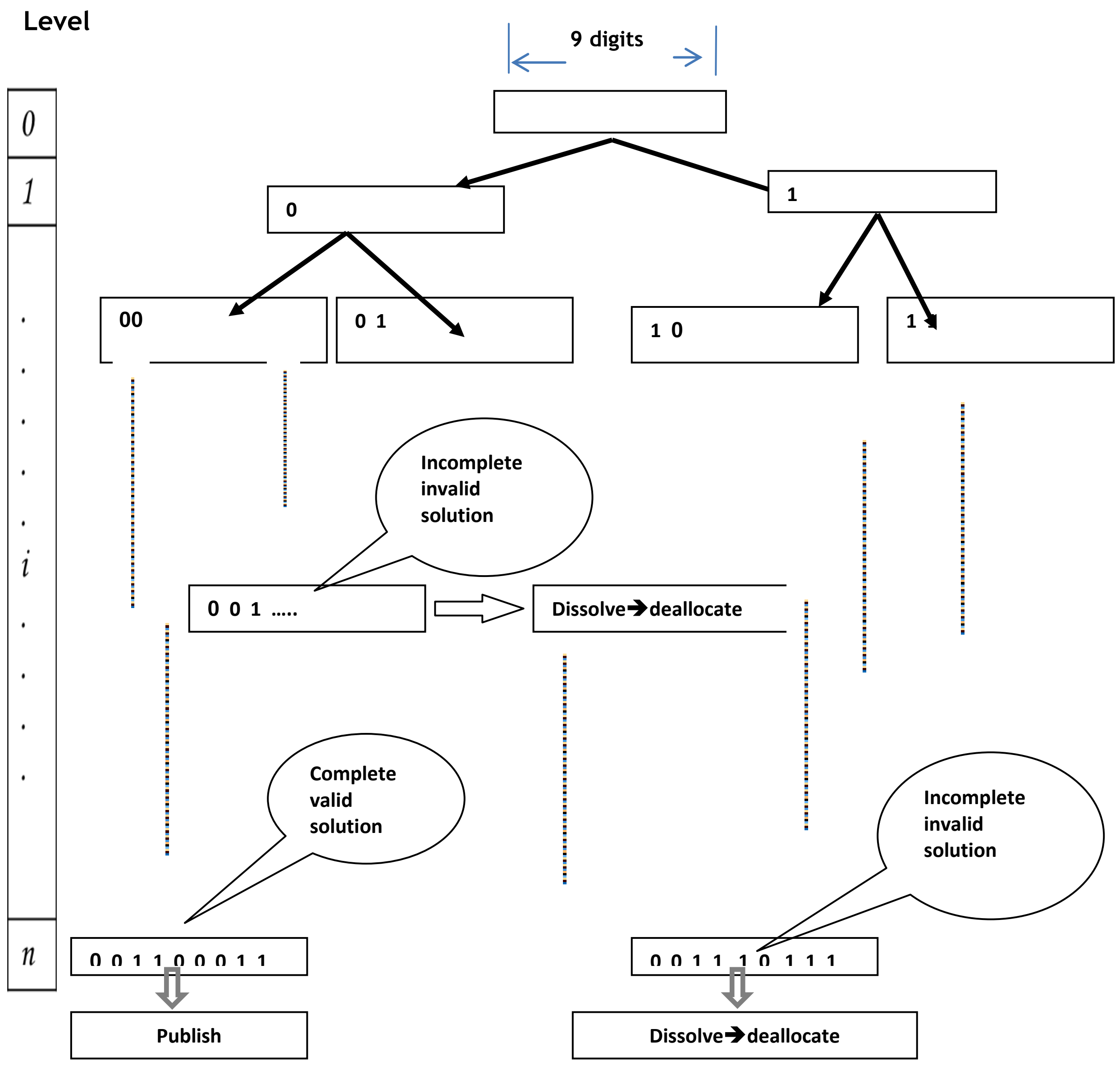

Fig 4: Illustration of solution development using the cloud based algorithm

The initial array size equals to the number of literals (literal is a variable or negation of variable), in our example the array size equals to 9 digits. There is an algorithm in the root instance which is equivalent to the set of rules in P systems. In level zero, as illustrated in figure 4, the algorithm creates two copies of the solution array then assigns zero (false) to the left most digit of the first copy and assigns one (true) to the left most digit of copy two. The assignment scheme is illustrated in figure 4. The root node in level zero is dissolved after division. Dissolution in our context means that the root instance is deallocated, e.g. the root instance becomes free and can be reassigned a new task. The same procedure is done with the two nodes in level one. The solution array can be checked for correctness, e.g., if the first clause is false then the result of all clauses is false, that is, short circuit evaluation can be used for performance enhancement. Only good solution will survive. In last level, we may have many solutions that constitute a pool of solutions that vary in their goodness. The question now is how we can get solutions in the last level. Each of these instances in the last level can write its solution in a shared accessed data server. An expert may choose among these solutions according to the underline problem that the SAT problem models. The instances could be lined up on queue to write their solutions. The number of nodes in the last level may be too much. The second question is how we can get in control of these solutions. A proposed solution is to attach a timer to every instance. The timer is activated only if the solution has two conditions; the solution array is both complete and valid. After a certain amount of time, the instance is automatically expired and dissolved. It should be taken into consideration that there is a proportional forward relation between the expiration time and the number of solutions written to the data server.

A simulation is implemented using multithreading, i.e., instead of separate instances on a cloud, we used threads. We solved SAT problem of size 11 literals, i.e., the generated search space equals 2^{11} solutions. The simulation assumes the worst case execution, that is to say all solutions survived to the last level of search space. The problem is solved in linear time rather than exponential one.

The algorithm deallocates instances with invalid incomplete solutions from the search space. But how can we filter solution and discard low goodness solutions? A heuristic function could be embedded in root node and consequently in all copied instances. This heuristic function is activated to work only when reaching the state that the solution array is valid and completed. The heuristic function deallocates instances with valid completed solution, yet with low goodness. The question now is: What do we mean by goodness? Goodness is a problem dependent term, for example, if the SAT problem models circuit design, then good solution may mean the solution that has a small number of ones.

This heuristic function will enhance the entire performance of the algorithm, namelyin the last level of the tree, only good valid completed solutions will wait in a queue to write their solutions, and an expert can select which one of them that will be more suitable. An illustration of a cloud instance contents is depicted in figure 5.

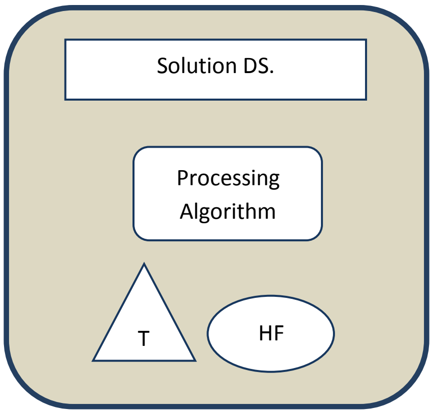

Fig 5: An illustration of an instance node, each node contains 4 objects: A solution data structure, a processing algorithm – that is responsible for node division and dissolution-, a Heuristic Filter (HF) for deallocation of low goodness completed solution, and finally, Timer (T) for automatic deallocation of all instances after predetermined amount of time.

6. CONCLUSIONS

P systems are a very interesting research area. P systems algorithms can solve NP-complete problems in practical amounts of time rather than exponential one. Till this moment, researchers have no idea about the exact tools to create a P system computer. The cloud based P system algorithm tries to develop an approximate implementation of p systems. The most important property of P systems is the massive parallelism, the exponential division of cells in linear or polynomial time and the independence between nodes. A P system generates all the search space like a brute force attack, i.e., generates an exponential one. This needs massive amounts of resources. After discovery of cloud computing, the resources are no longer an issue. The cloud based P system algorithm uses cloud resources and P systems computation model. The algorithm is applied on one of the most important NP-complete problems, i.e., the SAT problem. The proposed algorithm could be implemented to many other NP problems using the fact that every NP problem could be reduced to any NP-complete one.

7. REFERENCES

[1] Paun, G. 2006 Applications of membrane computing, Springer-Verlag, Berlin.

[2] Calude, C. and Paun, G. 2000 Computing with Cells and Atoms: An Introduction to Quantum, DNA and Membrane Computing, Romanian Academy, Bucharest, Romania.

[3] Petreska, B., Teuscher, C. 2004 A Hardware Membrane System. In Martin-Vide, C., Mauri, G., Paun, Gh., Rozenberg, G., Salomaa, A., eds. Membrane Computing. International Workshop, WMC2003, Tarragona, Spain, Revised Papers. Lecture Notes in Computer Science, 2933, Springer, Berlin.

[4] Ciobanu, G., Paraschiv, D. 2002 Membrane Software. A P System Simulator, Fundamenta Informaticae, Vol. 49, No. 1-3 ,pp. 61-66.

[5] Syropoulos, A., Mamatas, E., Allilomes, P., Sotiriades, K. 2003 A distributed simulation of P systems, Proceedings of the Workshop onMembrane Computing, pp. 455-460.

[6] Decastro, L. N. 2006 Fundamentals of Natural Computing: Basic Concepts, Algorithms, and Applications, CRC Press.

[7] Paun, G. 1998 Computing with membranes, Turku Center for Computer Science –TUCS No. 208.

[8] Paun, G. 2002 Computing with membranes: An introduction, Berlin: Springer.

[9] Paun, G. 2002 Membrane Computing: An Introduction, Springer.

[10] Cloud Computing, Academic Room. Retrieved 2012-8-8.

[11] Miller, M. 2008 Cloud Computing: Web-Based Applications That Change the Way You Work and Collaborate Online, Que.

[12] Cormen, T., Leiserson, C., Rivest, R. and Stein, C. 2009 Introduction to Algorithms, MIT Press.

[13] Engelbrecht, A.P. 2007 Computational Intelligence: An Introduction, Second Edition, John Wiley and Sons.

[14] Sakallah A. and Simon, L. 2011 Theory and Application of Satisfiability Testing, springer.