symmetry

MDPI

Article

Enhanced Membrane Computing Algorithm for SAT Problems Based on the Splitting Rule

Le Hao [1,*] and Jun Liu [2]

[1] School of Mathematics, Southwest Jiaotong University, Chengdu 611756, China
[2] School of Computing, Ulster University, Northern Ireland BT37 0QB, UK; j.liu@ulster.ac.uk
* Correspondence: haole2018@outlook.com; Tel.: +86-028-15708418678

Received: 13 October 2019; Accepted: 13 November 2019; Published: 15 November 2019

Abstract: Boolean propositional satisfiability (SAT) problem is one of the most widely studied NP-complete problems and plays an outstanding role in many domains. Membrane computing is a branch of natural computing which has been proven to solve NP problems in polynomial time with a parallel compute mode. This paper proposes a new algorithm for SAT problem which combines the traditional membrane computing algorithm of SAT problem with a classic simplification rule, the splitting rule, which can divide a clause set into two axisymmetric subsets, deal with them respectively and simultaneously, and obtain the solution of the original clause set with the symmetry of their solutions. The new algorithm is shown to be able to reduce the space complexity by distributing clauses with the splitting rule repeatedly, and also reduce both time and space complexity by executing one-literal rule and pure-literal rule as many times as possible.

Keywords: SAT problem; membrane computing; P system; splitting rule

1. Introduction

Boolean satisfiability problem, namely, SAT problem, is one of the most important problems of theoretical computer science. Its range of application includes multiple significance areas [1], such as mathematics, artificial intelligence, data mining, circuit design, etc., and it has attracted much attention since it was put forward. Since the 1960s, research has produced several models of SAT solvers, such as conflict driven clause learning [2], CDCL for short, in which its basic structure comes from the DPLL(Davis-Putnam-Logemann-Loveland) algorithm [3]. This model has made improvements to conflict analysis, clause learning, and some other aspects and has occupied the main battleground of SAT competitions. Representative solvers of this model include Mini-SAT [4], Lingeling [5], Chaff [6], and Glucose [7] all of which have received many achievements in international SAT competitions. With improvements of the SAT algorithm performance, many application examples can obtain satisfactory solutions in a given period of time, and SAT solvers are gradually being applied to more and more actual fields, such as circuit design verification [8,9] and cryptanalysis [10], however, as the first problem which has been proven as NP problem [11], the primary research direction of SAT problem is to reduce its computational complexity. Meanwhile, NP problems can transfer between each other in polynomial time, and therefore previous, current and future studies are also efforts to solve NP problems.

In recently decades, the research of SAT solvers has been mainly focused on three directions, complete solution algorithm, incomplete solution algorithm, and parallel solution algorithm. The membrane computing algorithm of SAT problems, which we are going to discuss, is a kind of parallel algorithm, which can solve any SAT problem in polynomial time but with exponential space occupation [12]. Among the three types of SAT solvers mentioned above, the complete solution algorithm is certain to obtain the solution of a given SAT problem, but it takes an unacceptable amount of time. By comparison, the incomplete solution algorithm uses less time, but it is not able

to ensure results. The disadvantages of these two algorithms can both be solved by the algorithm of membrane computing.

Membrane computing is a branch of natural computing, which was proposed in 1998 by professor Gheorghe Păun when he visited Finland [13]. Five years later, the Institute of Scientific Information listed it in the fast-growing frontier area of computational science. The membrane computing system, also known as P system, is a kind of distributed parallel computational model, with good computational performance by referring to and simulating the way cells, tissues, organs, or other biological structures process chemical substances, which has been provn to have the computing power of Turing machine, and a computing power to solve NP problems in polynomial time [14–18]. This characteristic has attracted significant attention from the scientific community who have promoted its development tremendously. To be specific, because membrane computation is performed at the cellular level, biochemical reactions and material transfer at the cellular level can be understood as computational processes. The cell membrane divides the cell into compartments, each compartment synchronously processes multiple resets of objects (corresponding to evolving compounds in the cell), the objects permeate through membranes, the membranes are dissolved, split, and produced, and their penetrability can also be changed. A series of transfers of a system is called a computation, and the calculation result is defined as the objects that appear in a particular membrane (also known as the output membrane) at termination. As a typical type of NP problem, solving SAT problems with membrane computing has a long history [19–23]. Although membrane computing has strong computational power, it is predicted that there is still an upper limit, and therefore developing a new model of membrane computing for solving SAT problems that simplifies algorithms' structure is also important, which is the aim of this paper.

In this paper, the traditional membrane computing algorithm of SAT problems is combined with a typical classic simplification rule, the splitting rule, to improve the algorithm's structure, from assigning values to all clauses in a membrane to dividing the given clauses into two parts, and deals with them respectively and simultaneously. Because this divide operation can be executed as many times as needed, it significantly reduces the space occupation of the algorithm. Meanwhile, it is obvious that a clause set dealt by the splitting rule must not include tautologies, one literals, and pure literals, and therefore it is indispensable to operate the following three simplified rules beforehand: tautology rule, one-literal rule, and pure-literal rule. The tautology rule needs only to be done once, at the beginning, and it does not make contributions to the simplification of the algorithm, however, the one-literal rule and pure-literal rule are two rules which can be used repeatedly and intensively, they can decrease the number of membrane divisions during the assignment process of the clause set, and can assign more than one value at one time, and therefore both space and time complexity of the algorithm is reduced.

The remainder of the paper is organized as follows: Section 2 provides some preliminary and review about P system and the traditional membrane computing algorithm for SAT, Section 3 proposes a new algorithm for SAT problem by combining the traditional membrane computing algorithm for SAT with splitting rule, in Section 4 an example is provided to illustrate the proposed algorithm, and the paper is concluded in Section 5.

2. Preliminaries

This section provides some preliminaries to be used in the present work.

2.1. Membrane Computing System (P System)

A membrane computing system [24] can be defined as $\prod = (O,\mu,w_i,R_i,i_0)$, of which:
O is a finite and nonempty alphabet of objects;
μ is a membrane structure made up by several membranes;
w_i denotes the character string inside the nth membrane in the initial state;
R_i is a finite set of the evolutionary rules which are carried out inside the ith membrane;

i_0 is the membrane which stores the final result.

2.2. Tradtional Membrane Computing Algorithm of SAT Problems

Using membrane computing system to solve SAT problem has decades of history. Figure 1 shows the process diagram of the traditional membrane computing algorithm of SAT problems [24].

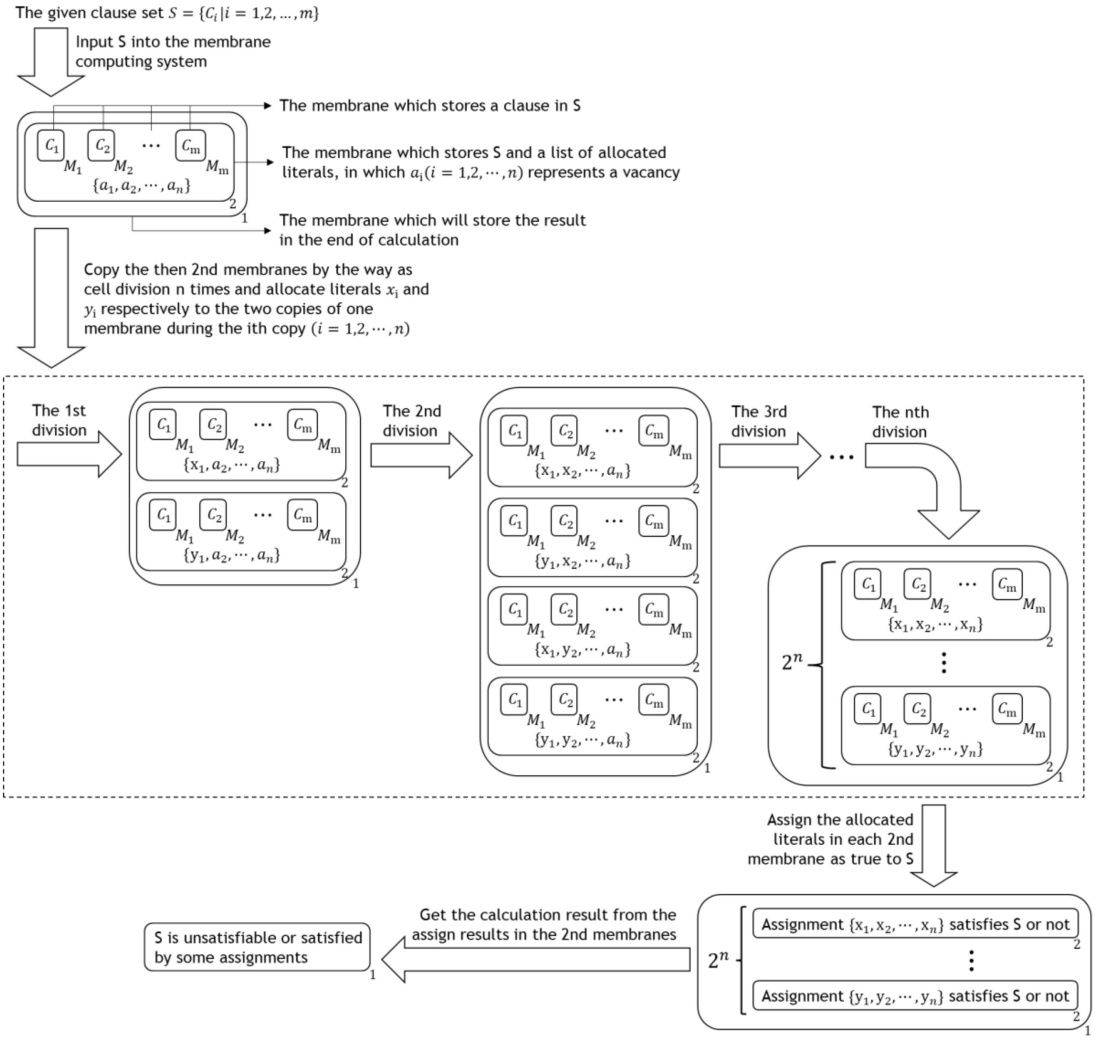

Figure 1. Process diagram of the traditional membrane computing algorithm of SAT problems, of which m is the number of given clauses, n is the number of the given clause set's atoms.

From the above diagram, shown in Figure 1, it is clear that this algorithm consists of the following two steps, except the input and output steps:

Step 1: Copy the, then, 2nd membranes by cell division n times and allocate literals x_i and y_i, respectively to the two copies of one membrane during the ith copy $(ir[1,n]\cap,)$, its time complexity is $O(1)$, and its once execution makes the space occupation of this algorithm twice as much as before.

Step 2: Assign the allocated literals in each 2nd membrane as true to the given clause set, its time complexity is $O(1)$, and since it is completed with sending each allocated literal into all membrane M_is which is inside the same 2nd membrane, the largest increase in space occupation of this algorithm during its execution process is $O(2^n n(m-1))$, and the space occupation of this algorithm will decrease to $O(2^n)$ after its execution process.

To sum up, the time complexity of this algorithm is O(n), and since the biggest initial space occupation of this algorithm is O(nm), the space complexity of this algorithm is $O(2^n nm) + O(2^n n(m-1)) = O(2^n nm)$.

2.3. Simplification Rules

Splitting Rule [25]: Assume a clause set S can be arranged into a form such as $[(A_1 \vee L) \wedge \cdots \wedge (A_n \vee L)] \wedge [(B_1 \vee \neg L) \wedge \cdots \wedge (B_m \vee \neg L)] \wedge R$, of which $A_i (i = 1, 2, \cdots, n)$ and $b_i (i = 1, 2, \cdots, m)$, R are clauses which exclude literals L and ¬L, then S is unsatisfiable if and only if $A_1 \wedge \cdots \wedge A_n \wedge R$ and $B_1 \wedge \cdots \wedge B_m \wedge R$ are unsatisfiable, A is an assignment which satisfies $A_1 \wedge \cdots \wedge A_n \wedge R$ $(B_1 \wedge \cdots \wedge B_m \wedge R)$ if and only if $A \cup \{\neg L = 1\} (A \cup \{L = 1\})$ is an assignment which satisfies S.

Tautology Rule [25]: Delete tautologies, namely the clauses which include complemental pairs of literals, from a clause set does not change if an assignment satisfies this clause set.

One-literal Rule [25]: Assume a clause set S includes a clause which only contains one literal L, then L is a single literal of S. Since one-literal clause can only be satisfied by assignments which assign its single literal as true, if S is empty after deleting the clauses contain S's single literals from S, S can be satisfied by any assignments which assign its single literals as true, or else S is satisfied by assignment A after deleting the clauses contain S's single literals from it and deleting literal the negations of S's single literals from all of its clauses if and only if S is satisfied by an assignment $A \cup \{$S's sin gle literals are all true$\}$.

Pure-literal Rule [25]: Assume a clause set S includes literal L but excludes literal ¬L, then L is a pure literal of S. Since the clause which contains pure literals can only be satisfied by assignments which assign the pure literals contained by it as true, if S is empty after deleting the clauses contain S's pure literals from S, S can be satisfied by any assignments which assign its pure literals as true, or else S is satisfied by assignment A after deleting the clauses contain S's pure literals from it if and only if S is satisfied by an assignment $A \cup \{$S's pure literals are all true$\}$.

3. Proposed New Algorithm

3.1. Definition

This part is the elements' definitions of the algorithm's membrane computing structure as follows:

(1) O: $\{x_i, y_i$, denote the literals of given clause set;

h, denotes the number of literals in a clause;

t_i, f_i, are the literal symbols of x_i and y_i;

g, denotes the number of literals' symbols;

a_i, denote the atoms of the given formulae;

T_i, F_i, are objects in the literal list of 2nd membrane;

c_i, e_i, are the transitive symbols of t_i and f_i;

p_i, denote the transport sign of membrane M_i;

z, denotes the number of membranes $M_i (i = 1, \ldots, m)$;

t_i, f_i, denote the assignments of literals x_i and y_i;

λ, δ, are the melting symbol of objects and membrane;

Y, N, a, b, c, d, p, q, s, are aided symbols$\}$.

(2) μ : the initial membrane structure is as Figure 2:

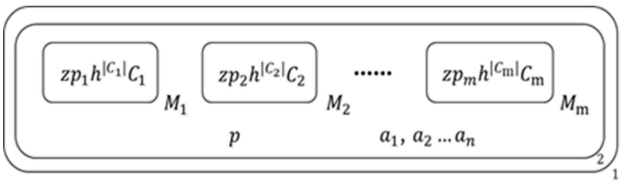

Figure 2. Initial membrane structure of the algorithm.

where C_j denote the jth clause in the given clause set, $\left|C_j\right|$ represents the number of literals in $C_j (j = 1, \ldots, m)$.

In addition, there's a literal list inside 2nd membrane which is empty in the initial status, it is used to store the literals which would be assigned as true in all assignments which satisfy the inside clause set of 2nd membrane and the given clause set.

(3) $w_1 = \lambda$;

$w_2 = p, a_i \ (i = 1, \ldots, n)$;

$w_{M_j} = z, p_j, h, x_i, y_i \ (i = 1, \ldots, n, \ j = 1, \ldots, m)$.

(4) $R_i (i = 1, \ 2, M_j)$ specific as described in the part of algorithm, and they are made up by three types of evolutionary rules:

(a) $[a]_i \rightarrow [b]_i [c]_i$, copy the membrane i into two copies when it or them contains object a, turn a in two copies into b and c respectively;

(b) $a \rightarrow b|c$ or $\neg c$, turn a into b when c is existent or inexistent;

(c) $a \rightarrow b\left(c, in_j \text{ or out}\right)$, turn a into b and c, meanwhile, send c into the membrane i or out the membrane which a was inside.

(5) $i_0 = 1$.

3.2. Compiling

As stated in the Introduction, the new membrane computing algorithm, proposed in this paper, is specifically based on the splitting rule and uses tautology rule, one-literal rule and pure-literal rule, and therefore for clarity this section compiles the simplified rules with membrane computing language, first, as follows:

(1) Tautology rule:

This rule is executed on the only 2nd membrane at the beginning of the calculation.

Pick tautologies from the clauses inside membrane M_is, and mark the membrane M_is which contain tautologies with a delete symbol, $r_1 = \left(zp_j h^2 x_i y_i \rightarrow s, 1\right)$;

Delete all contents inside the membrane M_is which are marked with a delete symbol:
$r_2 = (hx_i \rightarrow \lambda|s, 1)$
$r_3 = \left(hy_i \rightarrow \lambda|s, 1\right)$;

Delete the membrane M_is which are marked with a delete symbol, $r_4 = (s \rightarrow \delta, 1)$.

(2) One-literal rule:

This rule is executed on each 2nd membrane.

Pick the one-literal clauses from the clauses inside membrane M_is, mark the literals of these selected one-literal clauses with one-literal symbols t_i, f_i, send these symbols outside their original membranes.

$$\begin{aligned} r_1 &= (zp_j x_i \to x_i(zp_j gt_i, out) | \neg h^2, 4) \\ r_2 &= (zp_j y_i \to y_i(zp_j gf_i, out) | \neg h^2, 4) \ ; \\ r_3 &= (zp_j, out, 5) \end{aligned}$$

Then, judge if its inside clause set has complementary pairs of single literals with the one-literal symbols, if so, this clause set is unsatisfiable, so what we need to do is to delete this 2nd membrane and all its contents:

$$\begin{aligned} r_4 &= (g^2 t_i f_i \to N | \neg N, 1); \\ r_5 &= (zp_j \to (szp_j, inM_j) | N, 1) \\ r_6 &= (hx_i \to \lambda | s, 1) \\ r_7 &= (hy_i \to \lambda | s, 1) \\ r_8 &= (s \to \delta, 1) \\ r_9 &= (a_i \to \lambda | N, 1) \\ r_{10} &= (gt_i \to \lambda | N, 1) \\ r_{11} &= (gf_i \to \lambda | N, 1) \\ r_{12} &= (p \to \lambda | N, 1) \\ r_{13} &= (N \to \delta, 1) \end{aligned}$$

If not, since all assignments which satisfy the inside clause set assign these single literals as true, add these one-literal symbols into the literal list of this 2nd membrane:

$$\begin{aligned} r_{14} &= (g^2 t_i{}^2 \to gt_i, 2) \\ r_{15} &= (g^2 f_i{}^2 \to gf_i, 2) \\ r_{16} &= (ga_i t_i \to bt_i T_i | \neg t_i{}^2, 2) \\ r_{17} &= (ga_i f_i \to bf_i F_i | \neg f_i{}^2, 2) \end{aligned}$$

then package these one-literal symbols and copy this symbol pack into as many copies as membrane M_is inside this 2nd membrane, then send each copy into one membrane M_i:

$$\begin{aligned} r_{18} &= (bp \to bq[\,]_3 | \neg g, 1) \\ r_{19} &= (bt_i \to (t_i, in_3), 1) \\ r_{20} &= (bf_i, \to (f_i, in_3), 1) \\ r_{21} &= ([\,]_3 zp_j \to [\,]_3([\,\delta\,]_3 zp_j, inM_j) | \neg b, 1) \quad ; \\ r_{22} &= (q \to p(s, in_3) | \neg z, 1) \\ r_{23} &= (t_i \to \lambda | s, 1) \\ r_{24} &= (f_i \to \lambda | s, 1) \end{aligned}$$

Next execute following three operations respectively on the membrane M_is which contain different types of clauses:

For the membrane M_i whose inside clause includes some of these single literals, the inside clause can be satisfied by any assignment which assigns these single literals as true, so what needs to be done is delete it and all of its contents:

$$\begin{aligned} r_{25} &= (t_i zp_j hx_i \to s, 1) \\ r_{26} &= (f_i zp_j hy_i \to s, 1) \end{aligned} \quad ;$$

For the membrane M_i whose inside clause excludes these single literals, but includes some of their negations, the inside clause can be satisfied by an assignment which assigns these single literals as true if and only if it can still be satisfied by this assignment after deleting the negations of these single literals from it, so delete these negations:

$\begin{matrix} r_{27} = (t_i hy_i \rightarrow \lambda, 2) \\ r_{28} = (f_i hx_i \rightarrow \lambda, 2) \end{matrix}$, for the same reason, if the inside clause is built only by the negations of single literals, the clause set inside this 2nd membrane is unsatisfiable, delete this membrane and all of its contents:

$\begin{matrix} r_{29} = (zp_j \rightarrow \delta(N, out) \vert \neg h, 3) \\ r_{30} = (T_i \rightarrow \lambda \vert N, 1) \\ r_{31} = (F_i \rightarrow \lambda \vert N, 1) \\ r_{32} = (t_i \rightarrow \lambda \vert h, 3 \\ r_{33} = (f_i \rightarrow \lambda \vert h, 3 \end{matrix}$, or else delete the left one-literal symbols in the end:

For the membrane M_i whose inside clause excludes single literals and their negations, we can just delete the one-literal symbols from this membrane M_i.

(3) Pure-literal rule:

This rule is executed on each 2nd membrane as follows:

Send all literals in the inside clause set outside from their original membranes:

$\begin{matrix} r_1 = (\left[zp_j \right]_{M_i} \rightarrow [\ ]_{M_i} [c]_{M_i} (zp_j, out), 1) \\ r_2 = (h \rightarrow \lambda \vert c, 1) \\ r_3 = (c \rightarrow \delta, 1) \end{matrix}$, then delete the repetitive literals: $\begin{matrix} r_4 = (x_i^2 \rightarrow x_i, 1) \\ r_5 = (y_i^2 \rightarrow y_i, 1) \end{matrix}$,

finally delete the complemental pairs of literals: $\begin{matrix} r_6 = (x_i \rightarrow gt_i \vert \neg x_i^2, 1) \\ r_7 = (y_i \rightarrow gf_i \vert \neg y_i^2, 1) \\ r_8 = (g^2 t_i f_i \rightarrow \lambda, 1) \end{matrix}$

At this point, the left literals are all pure literals of the inside clause set, since all assignments which satisfy the inside clause set assign these pure literals as true, what needs to be done is that add the symbols of them into the literal list of this 2nd membrane: $\begin{matrix} r_9 = (ga_i t_i \rightarrow bt_i T_i, 2) \\ r_{10} = (ga_i f_i \rightarrow bf_i F_i, 2) \end{matrix}$, then package these one-literal symbols and copy these symbols into as many copies as membrane M_is inside this 2nd membrane, then send each copy into one membrane M_i:

$$\begin{matrix} r_{11} = (bp \rightarrow bq[\ ]_3 \vert \neg g, 1) \\ r_{12} = (bt_i \rightarrow (t_i, in_3), 1) \\ r_{13} = (bf_i \rightarrow (f_i, in_3), 1) \\ r_{14} = ([\ ]_3 zp_j \rightarrow [\ ]_3 ([\ \delta\ ]_3 zp_j, inM_j) \vert \neg b, 1) \\ r_{15} = (q \rightarrow p(s, in_3) \vert \neg z, 1) \\ r_{16} = (hx_i \rightarrow \lambda \vert s, 1) \\ r_{17} = (hy_i \rightarrow \lambda \vert s, 1) \\ r_{18} = (t_i \rightarrow \lambda \vert s, 1) \\ r_{19} = (f_i \rightarrow \lambda \vert s, 1) \\ r_{20} = (s \rightarrow \delta, 1) \end{matrix}$$

Next execute following three operations, respectively, on the membrane M_is which contain different types of clauses:

For the membrane M_i whose inside clause includes some of these pure literals, the inside clause can be satisfied by any assignment which assigns these pure literals as true, therefore what needs to be done is delete it and all of its contents: $\begin{matrix} r_{21} = (t_i zp_j hx_i \rightarrow s, 1) \\ r_{22} = (f_i zp_j hy_i \rightarrow s, 1) \end{matrix}$;

For the membrane M_i whose inside clause excludes single literals and their negations, we can just delete the one-literal symbols from this membrane M_i: $\begin{matrix} r_{23} = (t_i \rightarrow \lambda, 2) \\ r_{24} = (f_i \rightarrow \lambda, 2) \end{matrix}$.

(4) Splitting rule:

This rule is executed on each 2nd membrane whose inside clause set has no one-literal clause and pure-literal clause.

Splitting rule is supposed to be distributing the clauses in the given clause set into two parts which excludes a specified literal and its negation separately, and putting them into two new second membranes, but this operation is too complicated for the membrane computing program language, so we transpose the order of processes:

First, we copy the 2nd membrane and its contents into two copies and allocate a specified literal and its negation to two copies, respectively, the allocated literal of each copy is the literal which its negation and the clauses contain it should be excluded by the inside clause set of this copy, since all assignments which satisfy the inside clause set of one of two copies and the inside clause of the original 2nd membrane assign the allocated literal of this copy as true, add the symbol of the allocated literal into the literal list of each copy: $\begin{matrix} r_1 = \left(\mathrm{zp_j}, \mathrm{out}, 4\right) \\ \mathrm{r_2} = \left(\left[a_\mathrm{i} z^\mathrm{j}\right]_2 \to \left[\mathrm{t_i^j T_i}\right]_2 \left[\mathrm{f_i^j F_i}\right]_2 \middle| j \in Q^+, 1\right) \end{matrix}$, then send the literal symbols into each membrane M_i which inside two copies respectively:

$\mathrm{r_3} = (\mathrm{t_i p_j} \to (\mathrm{t_i zp_j}, \mathrm{inM_j}), 1;$

$\mathrm{r_4} = (\mathrm{f_i p_j} \to (\mathrm{f_i zp_j}, \mathrm{inM_j}), 1$

Next execute following three operations, respectively, on the membrane M_is which contain different types of clause.

For the membrane M_i whose inside clause includes the allocated literal of the 2nd membrane which contains it, this inside clause should be excluded by the inside clause set of this 2nd membrane, so what needs to be done is delete it and all of its contents: $\begin{matrix} \mathrm{r_5} = \left(\mathrm{t_i zp_j hx_i} \to \mathrm{s}, 1\right) \\ \mathrm{r_6} = \left(\mathrm{f_i zp_j hy_i} \to \mathrm{s}, 1\right) \\ \mathrm{r_7} = \left(\mathrm{hx_i} \to \lambda \middle| \mathrm{s}, 1\right) \\ \mathrm{r_8} = \left(\mathrm{hy_i} \to \lambda \middle| \mathrm{s}, 1\right) \\ \mathrm{r_9} = (\mathrm{s} \to \delta, 1) \end{matrix}$;

For the membrane M_i whose inside clause includes the negation of the allocated literal of the 2nd membrane which contains it, this negation should be excluded by the inside clause, so delete this negation: $\begin{matrix} \mathrm{r_{10}} = \left(\mathrm{t_i hy_i} \to \lambda, 2\right) \\ \mathrm{r_{11}} = (\mathrm{f_i hx_i} \to \lambda, 2) \end{matrix}$;

For the membrane M_i whose inside clause excludes the allocated literal of the 2nd membrane which contains this membrane M_i and its negation, we can just delete the literal symbol from this membrane M_i: $\begin{matrix} \mathrm{r_{12}} = (\mathrm{t_i} \to \lambda, 3) \\ \mathrm{r_{13}} = (\mathrm{f_i} \to \lambda, 3) \end{matrix}$;

Since these four program modules have a lot of overlap, we can concordance them to realize the algorithm of this paper. The flowing chart of the new algorithm is shown in Figure 3.

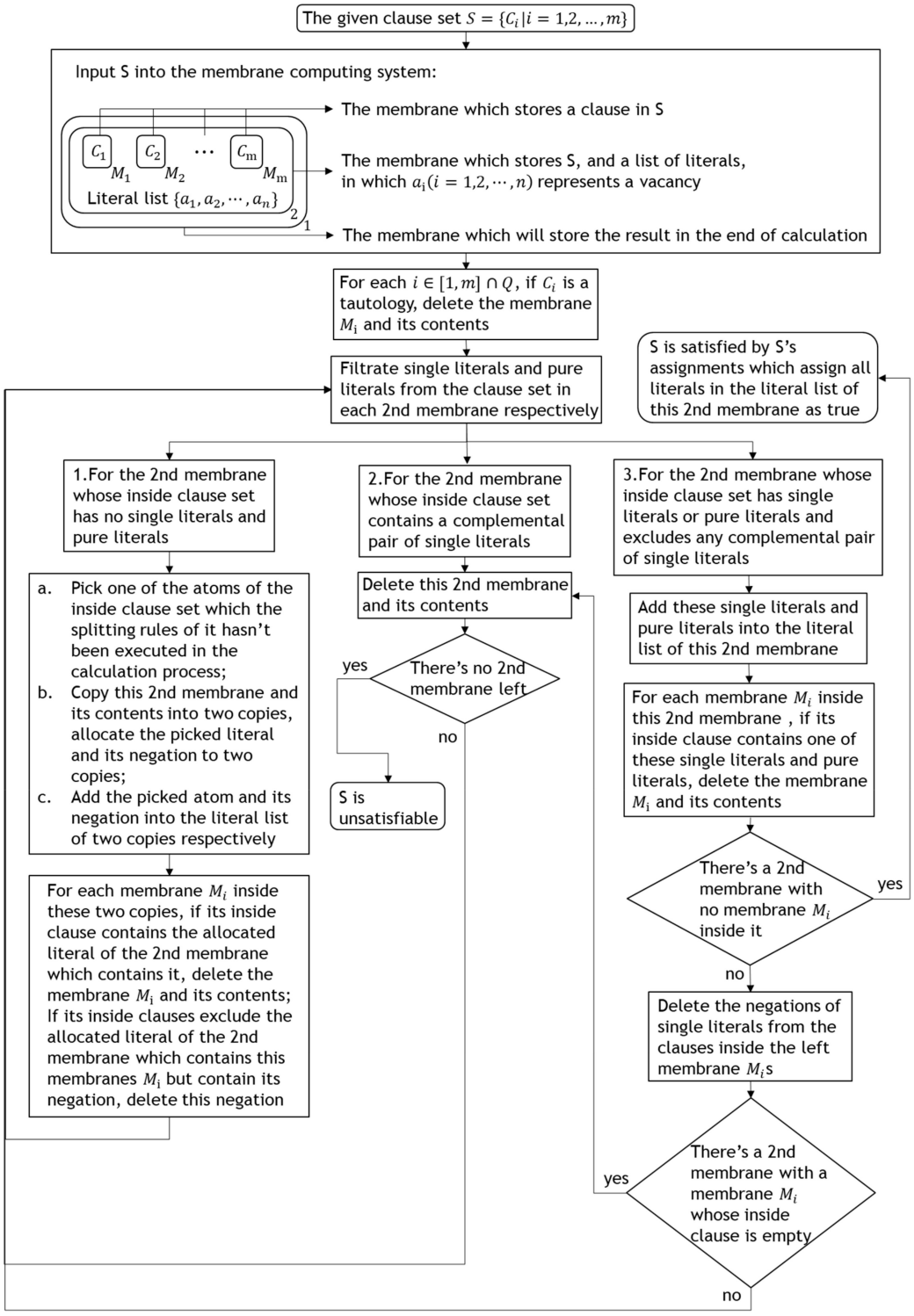

Figure 3. Flow chart of the algorithm.

From the above diagram (Figure 3), it is clear that this algorithm is completed by repeat executing the following two steps, in the ith $(1\text{th} < \text{n})$ repeat:

Step 1: Filtrate single literals and pure literals from the clause set in each 2nd membrane simultaneously, its time complexity is $O(1)$, the largest increase in space occupation of this algorithm during its execution process is no more than $O\left(2^{i-1}(m+1-i)(n+1-i)\right)$, and this increase will decrease to no more than $O\left(2^{i-1}(n+1-i)\right)$ after its execution process.

Step 2: Execute one of the following three substeps on each then 2nd membrane simultaneously based on the type of the 2nd membrane's inside clause set:

For the 2nd membrane whose inside clause set has no single literals and pure literals, copy this 2nd membrane into two copies and allocate an atom of this inside clause set and its negation, respectively, to the two copies, then in each copy, delete the clauses which contains its allocated literal from its inside clause set and the negation of its allocated literal from the clauses inside its inside clause set;

For the 2nd membrane whose inside clause set contains a complemental pair of single literals, delete this 2nd membrane and its contents;

For the 2nd membrane whose inside clause set has single literals or pure literals and excludes any complemental pair of single literals, assign these single literals and pure literals as true to its inside clause set.

The time complexity of this step is $O(1)$, and if the ith repeat is not the finial repeat, in the worst case, namely, in the case that all 2nd membranes' inside clause sets exclude single literals and pure literals, the largest increase in space occupation of this algorithm during the execution process of this step is no more than $O\left(2^{i-1}(m+1-i)(n+1-i)\right)+O\left(2^{i}(m-i)\right)$, and the space occupation of this algorithm will decrease to no more than $O\left(2^{i}(m-i)(n-i)\right)$ after its execution process; if not, all 2nd membranes' inside clause set has single literals or pure literals and excludes any complemental pair of single literals, the largest increase in space occupation of this algorithm during the execution process of this step is no more than $O\left(2^{i-1}(m+1-i)\right)$, and the space occupation of this algorithm will decrease to 0 after its execution process.

Since the upper bound of this algorithm's repeat time is Min(n,m), the time complexity of this algorithm is $O(\mathrm{Min}(n,m))$, the space complexity of this algorithm is:

$$\begin{aligned}&\mathrm{Max}_{ia[1,\mathrm{Min}(n,m)]\cap,}(\mathrm{Max}(O(2^{i-1}(m+1-i)(n+1-i))+\ O(2^{i-1}(m+1-i)(n+1-i))\\&+O(2^{i}(m-i))),O(2^{i-1}(m+1-i)(n+1-i))+O(2^{i-1}(m+1-i))(n+1-i)),\\&O(2^{i-1}(m+1-i)(n+1-i))+O(2^{i-1}(n+1-i))+O(2^{i-1}(m+1-i)))\\&=\mathrm{Max}_{ia[1,\mathrm{Min}(n,m)]\cap,}\left(O\left(2^{i}(m-i)(n-i)\right)\right)<O(2^{n}nm).\end{aligned}$$

It is clear that the new algorithm is more efficient than the traditional algorithm. To be noted, the new algorithm's computing complexity is worked out based on its worst case, which almost never happens, such as assuming that no pure-literal and one-literal clause exist before the final repeat, but the traditional algorithm's computing complexity is worked out based on the general change of the time and space which it occupies, and therefore, in practice, the difference between the two algorithm will be much more significant.

In order to describe the difference between the new algorithm's computing complexity and its general time and space occupation, work out the three situations of the time and space occupation of the step 2's execution on a 2nd membrane in the ith repeat, of which the ith repeat is not the finial repeat as follows:

If the 2nd membrane whose inside clause set has no single literals and pure literals, the time complexity is $O(1)$, the largest increase in space occupation of this 2nd membrane, during the execution process, is no more than $O((m+1-i)(n+1-i))+O(2(m-i))$, and the space occupation of this 2nd membrane will decrease to no more than $O(2(m-i)(n-i))$ after the execution process, the new algorithm's computing complexity is worked out based on this situation;

If the 2nd membrane whose inside clause set contains a complemental pair of single literals, the time complexity is $O(1)$, the largest increase in space occupation of this 2nd membrane during the execution process is 0, and the space occupation of this 2nd membrane will decrease to 0 after the execution process;

If the 2nd membrane whose inside clause set has single literals or pure literals and excludes any complemental pair of single literals, the time complexity is $O(1)$, the largest increase in space occupation of this 2nd membrane during the execution process is no more than $O((m-i)(n+1-i))$,

and the space occupation of this 2nd membrane will decrease to no more than $O((m-i)(n-i))$ after the execution process. In this situation, assume the inside clause set of this 2nd membrane has k ($k \le n+1-i$) single literals and pure literals, then k atoms of it have been assigned values simultaneously by this step.

Since the upper bound of the new algorithm's rest repeat time is no more than the biggest number of unassigned atoms of the then 2nd membranes' inside clause sets, the above step makes a contribution to reducing the repeat time of the new algorithm, namely reducing the time complexity of the new algorithm in the second and the third situations.

According to the above flow chart and program modules, we get the program of the object algorithm:

$$
\begin{aligned}
R_{M_i}: \quad & r_1 = (zp_j h^2 x_i y_i \to s, 1) \\
& r_2 = (t_i zp_j h x_i \to s, 1) \\
& r_3 = (f_i zp_j h y_i \to s, 1) \\
& r_4 = (hx_i \to \lambda|s, 1) \\
& r_5 = (hy_i \to \lambda|s, 1) \\
& r_6 = (t_i \to \lambda|s, 1) \\
& r_7 = (f_i \to \lambda|s, 1) \\
& r_8 = (c \to \lambda|s, 1) \\
& r_9 = (s \to \delta, 1) \\
& r_{10} = (t_i h y_i \to \lambda, 2) \\
& r_{11} = (f_i h x_i \to \lambda, 2) \\
& r_{12} = (t_i \to \lambda|h, 3) \\
& r_{13} = (f_i \to \lambda|h, 3) \\
& r_{14} = (zp_j \to \delta(N, out)|\neg h, 3) \\
& r_{15} = (h \to \lambda|c, 4) \\
& r_{16} = (c \to \delta|\neg s, 4) \\
& r_{17} = (x_i \to x_i(gt_i, out)|\neg h^2, 5) \\
& r_{18} = (y_i \to y_i(gf_i, out)|\neg h^2, 5)
\end{aligned}
$$

$$
\begin{aligned}
R_2: \quad & r_{19} = (g^2 t_i f_i \to N|\neg N, 1) \\
& r_{20} = (t_i p_j \to (t_i zp_j, inM_j)|\neg z, 1) \\
& r_{21} = (f_i p_j \to (f_i zp_j, inM_j)|\neg z, 1) \\
& r_{22} = (bp \to bq[\,]_3|\neg g, 1) \\
& r_{23} = (bt_i \to (t_i, in_3), 1) \\
& r_{24} = (bf_i \to (f_i, in_3), 1) \\
& r_{25} = ([\,]_3 zp_j \to [\,]_3([\,\delta\,]_3 zp_j, inM_j)|\neg b, 1) \\
& r_{26} = (q \to p(s, in_3)|\neg z, 1) \\
& r_{27} = (x_i \to c_i|\neg c_i, 1) \\
& r_{28} = (y_i \to e_i|\neg e_i, 1) \\
& r_{29} = (x_i \to \lambda|c_i, 1) \\
& r_{30} = (y_i \to \lambda|e_i, 1) \\
& r_{31} = (c_i e_i \to \lambda, 1) \\
& r_{32} = (c_i \to gt_i|\neg e_i, 1) \\
& r_{33} = (e_i \to gf_i|\neg c_i, 1) \\
& r_{34} = (zp_j \to (szp_j, inM_j)|N, 1) \\
& r_{35} = (a_i \to \lambda|N, 1) \\
& r_{36} = (gt_i \to \lambda|N, 1) \\
& r_{37} = (gf_i \to \lambda|N, 1) \\
& r_{38} = (p \to \lambda|N, 1) \\
& r_{39} = (T_i \to \lambda|N, 1) \\
& r_{40} = (F_i \to \lambda|N, 1) \\
& r_{41} = (N \to \delta, 1) \\
& r_{42} = (a_i \to \lambda|Y, 1) \\
& r_{43} = (Y \to \delta, 1) \\
& r_{44} = (gt_i \to \lambda|T_i, 1) \\
& r_{45} = (gf_i \to \lambda|F_i, 1) \\
& r_{46} = (ga_i t_i \to bt_i T_i|\neg T_i, 2) \\
& r_{47} = (ga_i f_i \to bf_i F_i|\neg F_i, 2) \\
& r_{48} = ([zp_j]_{M_i} \to [\,]_{M_i}[c]_{M_i}(zp_j, out), 2) \\
& r_{49} = ([a_i z^j]_2 \to [t_i^j T_i]_2 [f_i^j F_i]_2 | j \in Q^+, 3) \\
& r_{50} = (p \to q(d, out), 4)
\end{aligned}
$$

$$
\begin{aligned}
R_1: \quad & r_{51} = (d \to a|\neg a, 1) \\
& r_{52} = (d \to \lambda|a, 1) \\
& r_{53} = ([p]_2 \to [N]_2|Y, 1) \\
& r_{54} = ([q]_2 \to [N]_2|Y, 1) \\
& r_{55} = (a[q]_2 \to Y[Y]_2, 2) \\
& r_{56} = (Y \to \lambda, 3)
\end{aligned}
$$

The following is the explanation:

First, delete the membrane M_is which contain tautologies and their content with a delete symbol s: $r_1 = (zp_j h^2 x_i y_i \to s|\neg s, 1)$, this is how delete symbol s works in this algorithm:

$$
\begin{aligned}
& r_4 = (hx_i \to \lambda|s, 1) \\
& r_5 = (hy_i \to \lambda|s, 1) \\
& r_6 = (t_i \to \lambda|s, 1) \\
& r_7 = (f_i \to \lambda|s, 1) \\
& r_8 = (c \to \lambda|s, 1) \\
& r_9 = (s \to \delta, 1)
\end{aligned}
\quad ;
$$

Then filtrate single literals and pure literals from the clause set in each 2nd membrane respectively. First, execute the 22th rule to copy each membrane M_i and its contents into two copies, $r_{48} = \left(\left[\mathrm{zp_j}\right]_{M_i} \to [\,]_{M_i}[c]_{M_i}\left(\mathrm{zp_j}, out\right), 2\right)$, then for copy with mark c, release all literals of its inside clause to the 2nd membrane which contains it and generate literal symbols of all pure literals of this 2nd membrane's inside clause set with these released literals:

$$\begin{array}{l} \mathrm{r_{15}} = (h \to \lambda|\mathrm{c}, 4) \\ r_{16} = (\mathrm{c} \to \delta|\neg \mathrm{s}, 4) \\ \mathrm{r_{27}} = (\mathrm{x_i} \to \mathrm{c_i}|\neg \mathrm{c_i}, 1) \\ \mathrm{r_{28}} = \left(\mathrm{y_i} \to \mathrm{e_i}\middle|\neg \mathrm{e_i}, 1\right) \\ \mathrm{r_{29}} = (\mathrm{x_i} \to \lambda|\mathrm{c_i}, 1) \\ \mathrm{r_{30}} = \left(\mathrm{y_i} \to \lambda\middle|\mathrm{e_i}, 1\right) \\ \mathrm{r_{31}} = (\mathrm{c_i e_i} \to \lambda, 1) \\ \mathrm{r_{32}} = \left(\mathrm{c_i} \to \mathrm{gt_i}\middle|\neg \mathrm{e_i}, 1\right) \\ \mathrm{r_{33}} = \left(\mathrm{e_i} \to \mathrm{gf_i}\middle|\neg \mathrm{c_i}, 1\right) \end{array}$$

, and for the copy without mark c, if its inside clause is a one-literal clause, mark the literal of this inside clause with one-literal symbol and send the one-literal symbol outside:

$$\begin{array}{l} \mathrm{r_{17}} = \left(\mathrm{x_i} \to \mathrm{x_i}\left(\mathrm{gt_i}, \mathrm{out}\right)\middle|\neg \mathrm{h}^2, 5\right) \\ \mathrm{r_{18}} = \left(\mathrm{y_i} \to \mathrm{y_i}\left(\mathrm{gf_i}, \mathrm{out}\right)\middle|\neg \mathrm{h}^2, 5\right) \end{array};$$

Now execute following three operations on the 2nd membranes which contain different types of clause sets respectively:

(a) For the 2nd membrane whose inside clause set has no single literals and pure literals, pick one of the atoms of the inside clause set which the splitting rules of it hasn't been executed in the calculation process, and copy this 2nd membrane and its contents into two copies, allocate the picked literal and its negation to two copies, then add the picked atom and its negation into the literal list of two copies respectively, $\mathrm{r_{49}} = \left(\left[a_i z^j\right]_2 \to \left[\mathrm{t_i^j T_i}\right]_2\left[\mathrm{f_i^j F_i}\right]_2 \middle| j \in Q^+, 3\right)$, then for each membrane M_i inside these two copies, judge the type of its inside clause:

$$\begin{array}{l} \mathrm{r_{20}} = \left(\mathrm{t_i p_j} \to \left(\mathrm{t_i z p_j}, \mathrm{inM_j}\right)\middle|\neg z, 1\right) \\ \mathrm{r_{21}} = \left(\mathrm{f_i p_j} \to \left(\mathrm{f_i z p_j}, \mathrm{inM_j}\right)\middle|\neg z, 1\right) \end{array},$$

if its inside clause contains the allocated literal of the 2nd membrane which contains it, delete the membrane M_i and its contents, if its inside clauses exclude the allocated literal of the 2nd membrane which contains this membranes M_i, but contain its negation, delete this negation:

$$\begin{array}{l} \mathrm{r_2} = \left(\mathrm{t_i z p_j h x_i} \to \mathrm{s}, 1\right) \\ \mathrm{r_3} = \left(\mathrm{f_i z p_j h y_i} \to \mathrm{s}, 1\right) \\ \mathrm{r_{10}} = \left(\mathrm{t_i h y_i} \to \lambda, 2\right) \\ \mathrm{r_{11}} = (\mathrm{f_i h x_i} \to \lambda, 2) \\ \mathrm{r_{12}} = (\mathrm{t_i} \to \lambda|\mathrm{h}, 3) \\ \mathrm{r_{13}} = (\mathrm{f_i} \to \lambda|\mathrm{h}, 3) \end{array};$$

(b) For the 2nd membrane whose inside clause set contains a complemental pair of single literals, delete this 2nd membrane and its contents with a unsatisfiable symbol N: $\mathrm{r_{19}} = \left(\mathrm{g^2 t_i f_i} \to \mathrm{N}\middle|\neg \mathrm{N}, 1\right)$, this is how unsatisfiable symbol N works in this algorithm:

$$\begin{array}{l} \mathrm{r_{34}} = \left(\mathrm{zp_j} \to \left(\mathrm{szp_j}, \mathrm{inM_j}\right)\middle|\mathrm{N}, 1\right) \\ \mathrm{r_{35}} = (\mathrm{a_i} \to \lambda|\mathrm{N}, 1) \\ \mathrm{r_{36}} = \left(\mathrm{gt_i} \to \lambda\middle|\mathrm{N}, 1\right) \\ \mathrm{r_{37}} = \left(\mathrm{gf_i} \to \lambda\middle|\mathrm{N}, 1\right) \\ \mathrm{r_{38}} = (\mathrm{p} \to \lambda|\mathrm{N}, 1) \\ \mathrm{r_{39}} = (\mathrm{T_i} \to \lambda|\mathrm{N}, 1) \\ \mathrm{r_{40}} = (F_\mathrm{i} \to \lambda|\mathrm{N}, 1) \\ \mathrm{r_{41}} = (\mathrm{N} \to \delta, 1) \end{array},$$

if all 2nd membranes have been deleted, there's no rule can be executed, the membrane structure

has only the membrane which stores the final result left and this membrane is empty at this point, it means that S is unsatisfiable;

(c) For the 2nd membrane whose inside clause set has single literals or pure literals and excludes any complemental pair of single literals, first add these single literals and pure literals into the literal list of this 2nd membrane: $\begin{array}{l} \mathrm{r}_{44} = (\mathrm{gt_i} \to \lambda \left| T_\mathrm{i}, 1 \right.) \\ \mathrm{r}_{45} = (\mathrm{gf_i} \to \lambda \left| F_\mathrm{i}, 1 \right.) \\ \mathrm{r}_{46} = (\mathrm{ga_i t_i} \to \mathrm{bt_i} T_\mathrm{i} \left| \neg T_\mathrm{i}, 2 \right.) \\ \mathrm{r}_{47} = (\mathrm{ga_i f_i} \to \mathrm{bf_i} F_\mathrm{i} \left| \neg F_\mathrm{i}, 2 \right.) \end{array}$, then for each membrane M_i inside this 2nd membrane, judge the type of its inside clause: $\begin{array}{l} \mathrm{r}_{22} = (\mathrm{bp} \to \mathrm{bq}[\,]_3 \left| \neg \mathrm{g}, 1 \right.) \\ \mathrm{r}_{23} = (\mathrm{bt_i} \to (\mathrm{t_i}, \mathrm{in}_3), 1) \\ \mathrm{r}_{24} = (\mathrm{bf_i} \to (\mathrm{f_i}, \mathrm{in}_3), 1) \\ \mathrm{r}_{25} = \left([\,]_3 \mathrm{zp_j} \to [\,]_3 ([\,\delta\,]_3 \mathrm{zp_j}, \mathrm{inM_j}) \left| \neg \mathrm{b}, 1 \right. \right) \\ \mathrm{r}_{26} = (\mathrm{q} \to \mathrm{p}(\mathrm{s}, \mathrm{in}_3) \left| \neg \mathrm{z}, 1 \right.) \end{array}$, if its inside clause contains one of these single literals and pure literals, delete the membrane M_i and its contents: $\begin{array}{l} r_2 = (t_i z p_j h x_i \to s, 1) \\ r_3 = (f_i z p_j h y_i \to s, 1) \end{array}$, or else delete the negations of single literals from the clauses inside the left membrane M_is: $\begin{array}{l} \mathrm{r}_{10} = (\mathrm{t_i hy_i} \to \lambda, 2) \\ \mathrm{r}_{11} = (\mathrm{f_i hx_i} \to \lambda, 2) \\ \mathrm{r}_{12} = (\mathrm{t_i} \to \lambda \left| \mathrm{h}, 3 \right.) \\ \mathrm{r}_{13} = (\mathrm{f_i} \to \lambda \left| \mathrm{h}, 3 \right.) \end{array}$, if there's a membrane M_i whose inside clause is empty at this point, delete this 2nd membrane and its contents: $\mathrm{r}_{14} = (\mathrm{zp_j} \to \delta(\mathrm{N}, \mathrm{out}) \left| \neg h, 3 \right.)$.

If there's a 2nd membrane in which all membrane M_is have been deleted, S is satisfied by S's assignments which assign all literals in the literal list of this 2nd membrane as true, so what needs to be done is that pick a 2nd membrane like this, and delete all 2nd membrane and their contents except the literal list of the picked 2nd membrane $\begin{array}{l} \mathrm{r}_{50} = (\mathrm{p} \to \mathrm{q}(\mathrm{d}, out), 4) \\ r_{51} = (d \to a | \neg a, 1) \\ r_{52} = (d \to \lambda | a, 1) \\ r_{53} = \left([p]_2 \to [N]_2 \left| Y, 1 \right. \right) \\ r_{54} = \left([q]_2 \to [N]_2 \left| Y, 1 \right. \right) \\ r_{55} = \left(a[q]_2 \to Y[Y]_2, 2 \right) \\ r_{56} = (Y \to \lambda, 3) \\ \mathrm{r}_{42} = (\mathrm{a_i} \to \lambda | \mathrm{Y}, 1) \\ \mathrm{r}_{43} = (\mathrm{Y} \to \delta, 1) \end{array}$.

4. Example Illustration

In this section, we use an example to show how to solve a SAT problem with this new algorithm, we use clause set $\{x_1, y_1x_2y_3, x_3x_4, y_1y_3y_4\}$:

Figure 4 show the initial membrane structure of the given clause set.

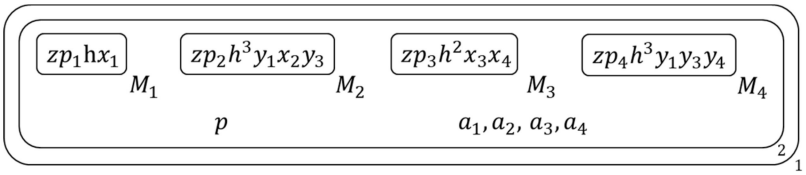

Figure 4. The initial membrane structure of the given clause set.

Delete tautologies from the given clause set (r_1): $\{x_1, y_1x_2y_3, x_3x_4, y_1y_3y_4\}$ includes no tautology.

Filtrate single literals and pure literals from the clause set in each 2nd membrane respectively:

1. Copy each membrane M_i and its contents into two copies (r_{48}), Figure 5 shows the membrane structure of this time.

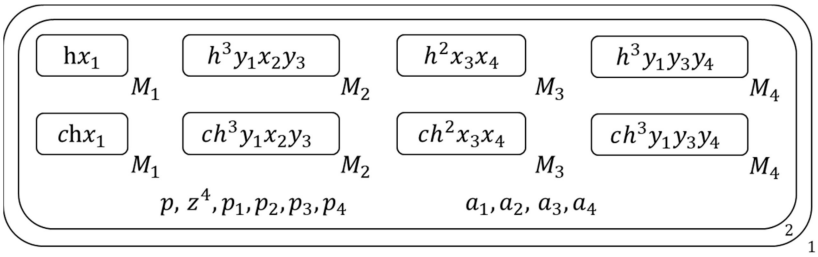

Figure 5. The membrane structure during calculation (1).

2. Then for copy with mark c, release all literals of its inside clause to the 2nd membrane which contains it and generate literal symbols of all pure literals of this 2nd membrane's inside clause set with these released literals $\left(r_{(15,16,27-33)}\right)$, and for the copy without mark c, if its inside clause is a one-literal clause, mark the literal of this inside clause with one-literal symbol and send the one-literal symbol outside $\left(r_{(17,18)}\right)$, Figure 6 shows the membrane structure of this time.

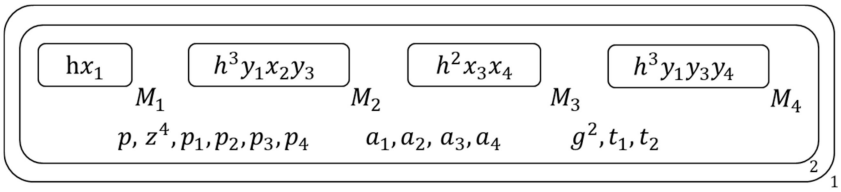

Figure 6. The membrane structure during calculation (2).

Execute the following operations on the 2nd membrane bases on the type of its inside clause set:

1. Add these single literals and pure literals into the literal list of this 2nd membrane $\left(r_{(44-47)}\right)$, Figure 7 shows the membrane structure of this time.

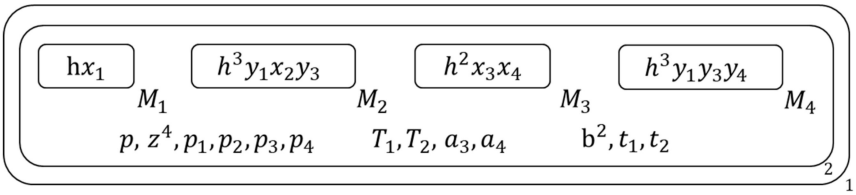

Figure 7. The membrane structure during calculation (3).

2. Then for each membrane M_i inside this 2nd membrane, judge the type of its inside clause $\left(r_{(22-26)}\right)$, Figure 8 shows the membrane structure of this time.

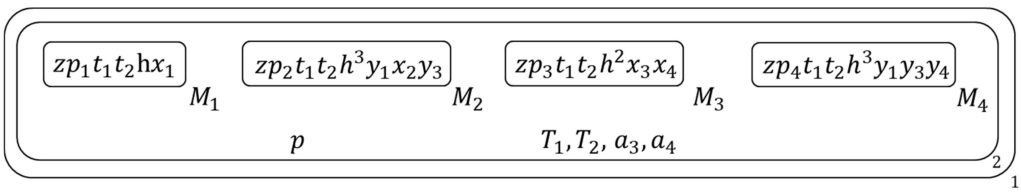

Figure 8. The membrane structure during calculation (4).

At this time, two of the S's atoms have been assigned, the membrane structures of the tradition algorithm which reach the same assign effect is as Figure 9.

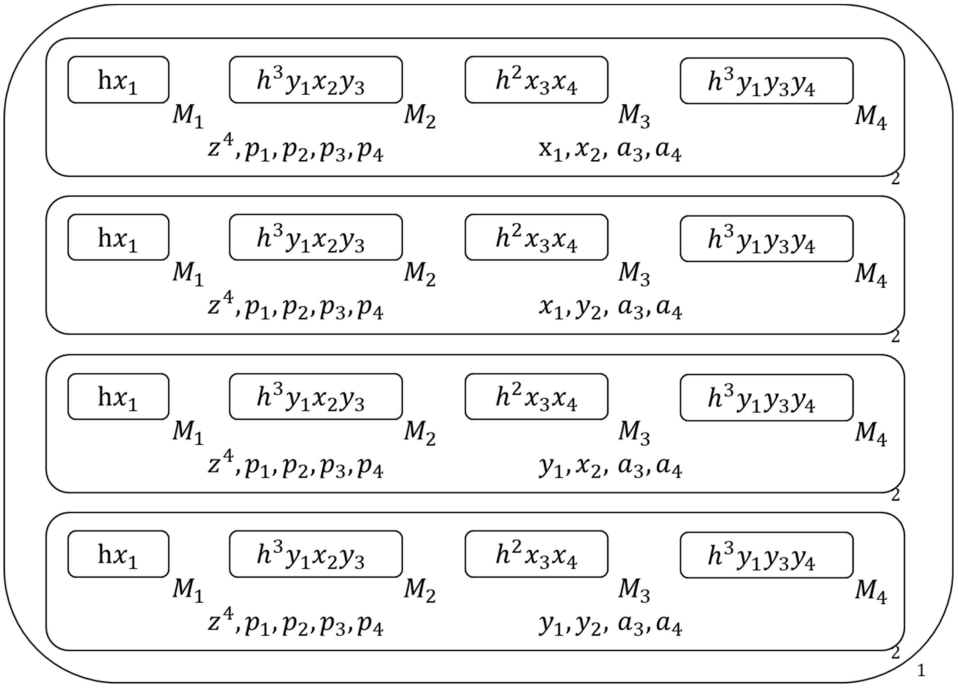

Figure 9. The membrane structure during calculation (5).

3. Next execute following three operations respectively on the membrane M_is which contain different types of clauses $\left(r_{(2,3,10-13)}\right)$, Figure 10 shows the membrane structure of this time.

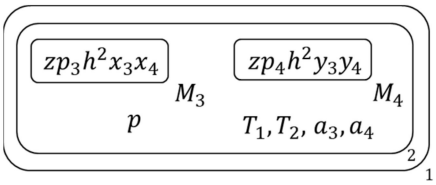

Figure 10. The membrane structure during calculation (6).

Filtrate single literals and pure literals from the clause set in each 2nd membrane respectively:

1. Copy each membrane M_i and its contents into two copies (r_{48}), Figure 11 shows the membrane structure of this time.

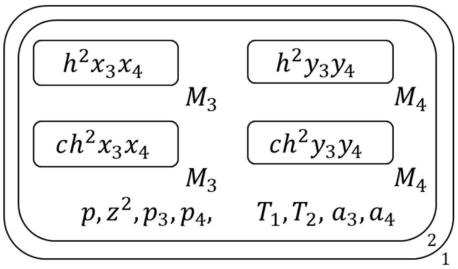

Figure 11. The membrane structure during calculation (7).

2. Then for the copy with mark c, release all literals of its inside clause to the 2nd membrane which contains it and generate literal symbols of all pure literals of this 2nd membrane's inside clause

set with these released literals $\left(r_{(15,16,27-33)}\right)$, and for the copy without mark c, if its inside clause is a one-literal clause, mark the literal of this inside clause with one-literal symbol and send the one-literal symbol outside $\left(r_{(17,18)}\right)$, Figure 12 shows the membrane structure of this time.

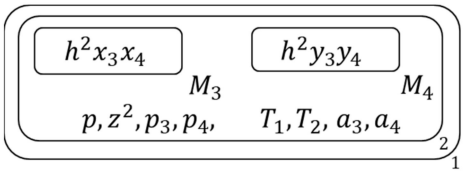

Figure 12. The membrane structure during calculation (8).

Execute the following operations on the 2nd membrane bases on the type of its inside clause set:

1. Pick one of the atoms of the inside clause set which the splitting rules of it hasn't been executed in the calculation process, and copy this 2nd membrane and its contents into two copies, allocate the picked literal and its negation to two copies, then add the picked atom and its negation into the literal list of two copies respectively (r_{49}), Figure 13 shows the membrane structure of this time.

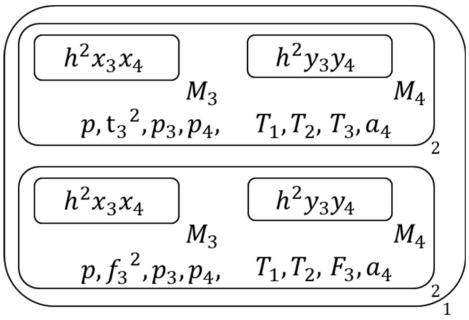

Figure 13. The membrane structure during calculation (9).

At this time, two of the S's atoms have been assigned, the membrane structures of the tradition algorithm which reach the same assign effect is as Figure 14.

2. Then for each membrane M_i inside these two copies, judge the type of its inside clause $\left(r_{(20,21)}\right)$, Figure 15 shows the membrane structure of this time.
3. Next execute following three operations respectively on the membrane M_is which contain different types of clauses $\left(r_{(2,3,10-13)}\right)$, Figure 16 shows the membrane structure of this time.

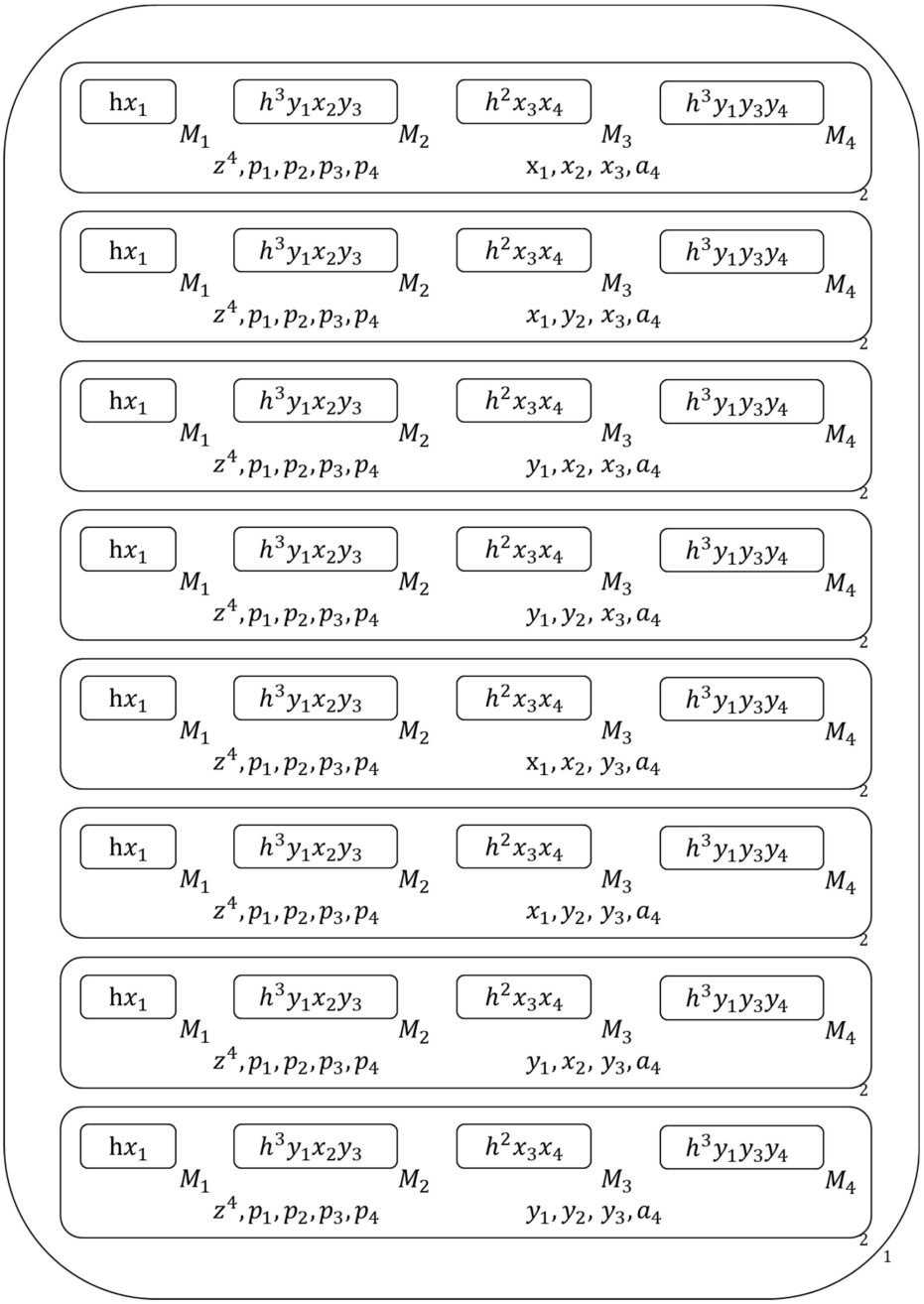

Figure 14. The membrane structure during calculation (10).

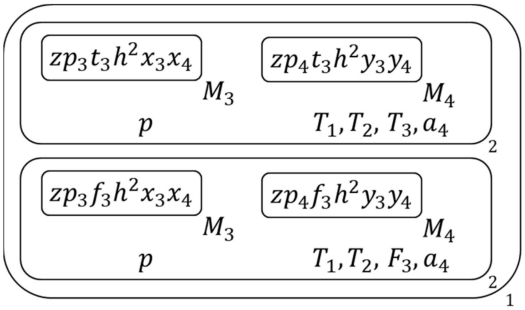

Figure 15. The membrane structure during calculation (11).

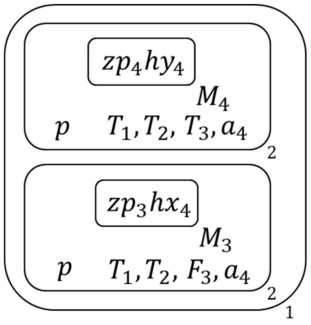

Figure 16. The membrane structure during calculation (12).

Filtrate single literals and pure literals from the clause set in each 2nd membrane respectively:

1. Copy each membrane M_i and its contents into two copies (r_{48}), Figure 17 shows the membrane structure of this time.

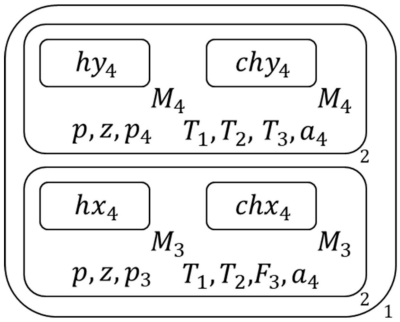

Figure 17. The membrane structure during calculation (13).

2. Then for the copy with mark c, release all literals of its inside clause to the 2nd membrane which contains it and generate literal symbols of all pure literals of this 2nd membrane's inside clause set with these released literals $\left(r_{(15,16,27-33)}\right)$, and for the copy without mark c, if its inside clause is a one-literal clause, mark the literal of this inside clause with one-literal symbol and send the one-literal symbol outside $\left(r_{(17,18)}\right)$, Figure 18 shows the membrane structure of this time.

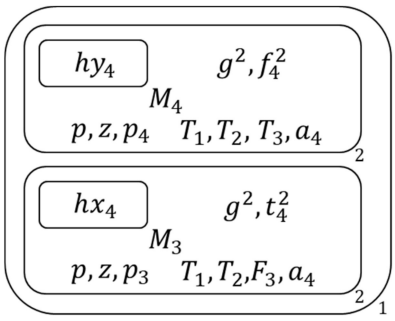

Figure 18. The membrane structure during calculation (14).

Execute the following operations on the 2nd membrane bases on the type of its inside clause set:

1. Add these single literals and pure literals into the literal list of this 2nd membrane $\left(r_{(44-47)}\right)$, then for each membrane M_i inside this 2nd membrane, judge the type of its inside clause $\left(r_{(22-26)}\right)$, Figure 19 shows the membrane structure of this time.

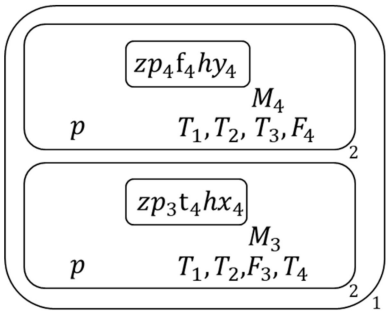

Figure 19. The membrane structure during calculation (15).

At this time, two of the S's atoms have been assigned, the membrane structures of the tradition algorithm which reach the same assign effect is as Figure 20.

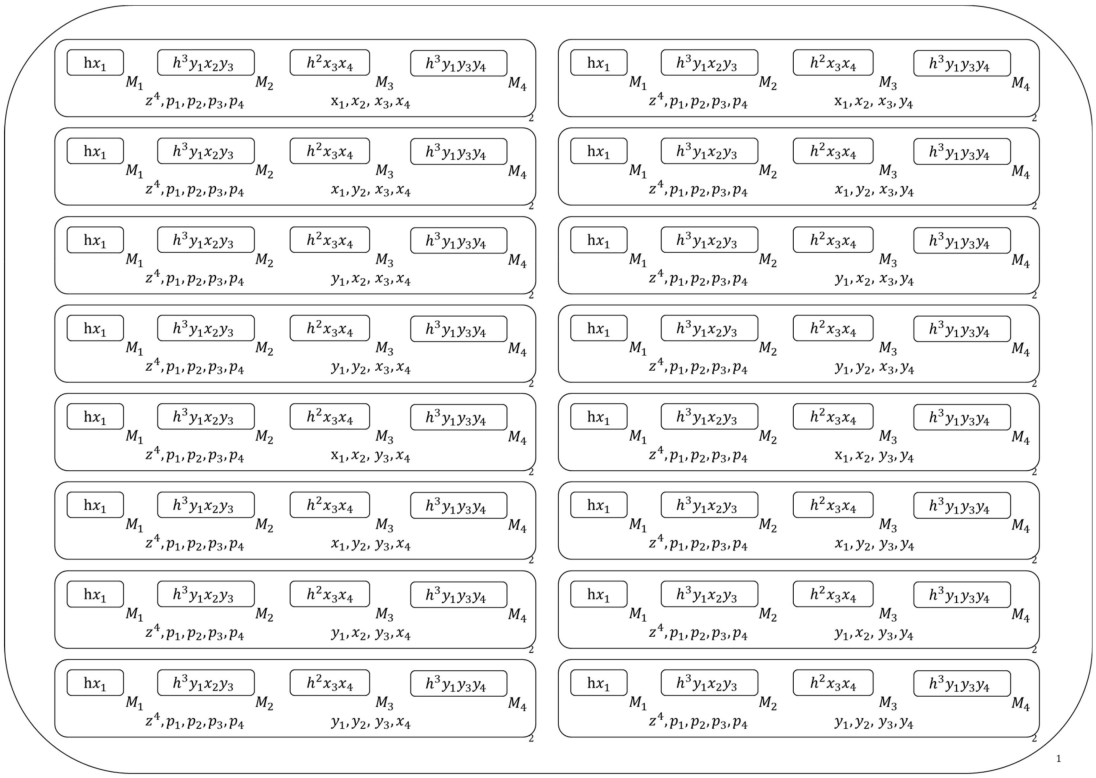

Figure 20. The membrane structure during calculation (16).

2. Next execute following three operations respectively on the membrane M_is which contain different types of clauses $\left(r_{(2,3,10-13)}\right)$, Figure 21 shows the membrane structure of this time.

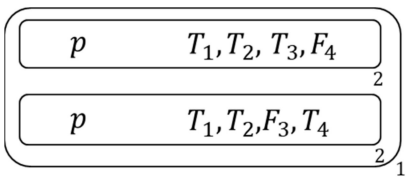

Figure 21. The membrane structure during calculation (17).

3. Pick a 2nd membrane like this, and delete all 2nd membrane and their contents except the literal list of the picked 2nd membrane $\left(r_{(50\text{–}56,42,43)}\right)$, Figure 22 shows the membrane structure of this time.

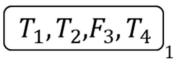

Figure 22. The membrane structure during calculation (18).

So we get the calculation result: the given clause set $\{x_1, y_1x_2y_3, x_3x_4, y_1y_3y_4\}$ is satisfied by assignment $\{x_1 = 1; x_2 = 1; y_3 = 1; x_4 = 1\}$.

5. Conclusions

Membrane computing is a type of natural computing, similar to the neural algorithm which has been widely used, however, unlike the neural algorithm, membrane computing is still not a reality, because its infinite parallel computing mode cannot be realized under current technical conditions, Nevertheless, calculating it as the way living bodies deal with data is desired, and therefore work towards it has never stopped as it is still just a fantasy. Even if it is realized and it works as well as our body cells, simplifying the algorithms based on it is still necessary because of their unimaginable but still limited computational power and the predictable crazy growing scale and needs of data treatment in the future. The work in this paper combines the membrane computing system with the traditional SAT problem simplification rules and obtains a fairly good effect, which has certain promotion and reference significance for the research field of membrane computing model and SAT problem solving model, and therefore has research value for related research fields. Meanwhile, this research indicates a feasible direction to improve membrane computing systems for different practice problems, which is to optimize the theoretical basis of this problem based on the character of P system. In addition, with respect to this research, in order to realize the membrane computing system, the strategy could be changed. Because membrane computing is a compute process which imitates the way cells work, the use of cells to simulate the compute process of SAT problems altogether should be considered. Although this has not been considered in this paper because of a lack of biological knowledge, it is a method that should be undertaken.

Author Contributions: Conceptualization, L.H.; methodology, L.H.; validation, L.H.; formal analysis, L.H.; investigation, L.H.; data curation, L.H.; writing—original draft preparation, L.H.; writing—review and editing, J.L.

Funding: This work was supported by the National Natural Science Foundation of China(Grant No.61673320), and by the Fundamental Research Funds for the Central Universities (Grant No.2682018ZT10, No.2682018CX59).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Kautz, H.; Selman, B. The State of SAT. *Discrete Appl. Math.* **2007**, *155*, 1514–1524. [CrossRef]
2. Marques-Silva, J.; Lynce, I.; Malik, S. Conflict-driven clause learning SAT solvers. In *Handbook of Satisfiability*; Biere, A., Heule, M., Van Maaren, H., Walsh, T., Eds.; IOS Press: Amsterdam, The Netherlands, 2009; Volume 185, pp. 131–153.
3. Davis, M.; Putnam, H. A Computing Procedure for Quantification Theory. *J. ACM* **1960**, *7*, 201–215. [CrossRef]
4. Sorensson, N.; Een, N. MiniSat—A SAT solver with conflict-clause minimization. In Proceedings of the SAT-05 8th International Conference on Theory and Applications of Satisfiability Testing, St Andrews, UK, 19–23 June 2005.
5. Biere, A. MiniSat, cadical, lingeling, plingeling, treengeling and yalsat Entering the SAT Competition 2018. In Proceedings of the SAT Competition 2018: Solver and Benchmark Descriptions; 2018.
6. Moskewicz, M.W.; Madigan, C.F.; Zhao, Y.; Zhang, L.; Malik, S. Chaff: Engineering an efficient SAT solver. In Proceedings of the 38th annual Design Automation Conference, Las Vegas, NV, USA, 22 June 2001.

7. Simon, L.; Audemard, G. Predicting Learnt Clauses Quality in Modern SAT Solver. In Proceedings of the 21th International Joint Conference on Artificial Intelligence, Pasadena, CA, USA, 11–17 July 2009.
8. Juretus, K.; Savidis, I. Importance of Multi-parameter SAT Attack Exploration for Integrated Circuit Security. In Proceedings of the 2018 IEEE Asia Pacific Conference on Circuits and Systems, Chengdu, China, 26–30 October 2018.
9. Liu, C.; Zhang, L.; He, X.; Guo, Y. Analysis of SET Reconvergence and Hardening in the Combinational Circuit Using a SAT-Based Method. *IEEE Access* **2018**, *6*, 48740–48746. [CrossRef]
10. Efficient Methods for Conversion and Solution of Sparse Systems of Low-Degree Multivariate Polynomials over GF(2) Via Sat-Solvers. Available online: http://eprint.iacr.org/2007/024 (accessed on 25 January 2007).
11. Cook, S.A. The complexity of theorem-proving procedures. In Proceedings of the Third ACM Symposium on Theory of Computing, Shaker Heights, OH, USA, 3–5 May 1971.
12. Martins, R.; Manquinho, V.; Lynce, I. An overview of parallel SAT solving. *Constraints* **2012**, *17*, 304–347. [CrossRef]
13. Păun, G. Computing with Membranes. *J. Comput. Syst. Sci.* **2000**, *61*, 108–143. [CrossRef]
14. Gutiérrez-Naranjo, M.A.; Pérez-Jiménez, M.J.; Riscos-Núñez, A.; Romero-Campero, F.J. On the efficiency of cell-like and tissue-like recognizing membrane systems. *Int. J. Intell. Syst.* **2009**, *24*, 747–765. [CrossRef]
15. Gutiérrez-Naranjo, M.A.; Pérez-Jiménez, M.J.; Riscos-Núñez, A.; Romero-Campero, F.J. On the power of dissolution in P systems with active membranes. In Proceedings of the International Workshop on Membrane Computing, Vienna, Austria, 18–21 July 2005; pp. 224–240.
16. Macías-Ramos, L.F.; Pérez-Jiménez, M.J.; Riscos-Núñez, A.; Rius-Font, M. The efficiency of tissue P systems with cell separation relies on the environment. In Proceedings of the International Conference on Membrane Computing, Budapest, Hungary, 28–31 August 2012; pp. 243–256.
17. Pan, L.; Pérez-Jiménez, M.J. Computational complexity of tissue–like P systems. *J. Complex.* **2010**, *26*, 296–315. [CrossRef]
18. Pérez-Jiménez, M.J.; Riscos-Núñez, A.; Rius-Font, M.; Valencia-Cabrera, L. The relevance of the environment on the efficiency of tissue P systems. In Proceedings of the International Conference on Membrane Computing Chisinau, Chisinau, Moldova, 20–23 August 2013; pp. 308–321.
19. Manca, V. DNA and membrane algorithms for SAT. *Fund. Inform.* **2002**, *49*, 205–221.
20. Ciobanu, G.; Păun, G.; Pérez-Jiménez, M.J. *Applications of Membrane Computing*; Springer: Berlin, Germany, 2006.
21. Ishii, K.; Fujiwara, A.; Tagawa, H. Asynchronous P systems for SAT and Hamiltonian cycle problem. In Proceedings of the 2010 Second World Congress on Nature and Biologically Inspired Computing, Fukuoka, Japan, 15–17 December 2010.
22. Song, T.; Macías-Ramosb, L.F.; Pan, L.; Pérez-Jiménez, M.J. Time-free solution to SAT problem using P systems with active membranes. *Theor. Comput. Sci.* **2014**, *529*, 61–68. [CrossRef]
23. Adorna, H.N.; Pan, L.; Song, B. On Distributed Solution to SAT by Membrane Computing. *Int. J. Comp. Commun.* **2018**, *13*, 303–322. [CrossRef]
24. Păun, G. *Membrane Computing: An Introduction*; Springer: Berlin, Germany, 2002.
25. Liu, X.H. *Automatic Reasoning Based on The Resolution Method*; Science Press: Beijing, Chinese, 1994.