Solving the SAT problem using a DNA computing algorithm based on ligase chain reaction

Xiaolong Wang *, Zhenmin Bao, Jingjie Hu, Shi Wang, Aibin Zhan
Department of Biotechnology, Ocean University of China, Qingdao 266003, People’s Republic of China
Received 4 April 2007; received in revised form 12 August 2007; accepted 17 August 2007

Abstract

A new DNA computing algorithm based on a ligase chain reaction is demonstrated to solve an SAT problem. The proposed DNA algorithm can solve an $n$-variable $m$-clause SAT problem in $m$ steps and the computation time required is $O(3^m + n)$. Instead of generating the full-solution DNA library, we start with an empty test tube and then generate solutions that partially satisfy the SAT formula. These partial solutions are then extended step by step by the ligation of new variables using Taq DNA ligase. Correct strands are amplified and false strands are pruned by a ligase chain reaction (LCR) as soon as they fail to satisfy the conditions. If we score and sort the clauses, we can use this algorithm to markedly reduce the number of DNA strands required throughout the computing process. In a computer simulation, the maximum number of DNA strands required was $2^{0.48n}$ when $n = 50$, and the exponent ratio varied inversely with the number of variables $n$ and the clause/variable ratio $m/n$. This algorithm is highly space-efficient and error-tolerant compared to conventional brute-force searching, and thus can be scaled-up to solve large and hard SAT problems.

Keywords: DNA computing; SAT problem; Ligase chain reaction; Space complexity; Time complexity

1. Introduction

DNA computing is a newly emerging interdisciplinary science that uses molecular biotechnologies to solve problems in computer science or mathematics. In their pioneering studies, Adleman and Lipton solved combinatorial problems [Hamilton path problem (HPP) (Adleman, 1994) and satisfiability problem (SAT) (Lipton, 1995)], using DNA computing algorithms based on a brute-force search. At the beginning of computation, they constructed a DNA pool that contained the full-solution space, and then extracted correct answers and/or eliminated false ones from the pool step by step. Thus, the number of distinct DNA strands required in the initial data pool grows exponentially with the size of the problem, and eventually swamps the DNA data storage for large problems, which makes molecular computation impractical from the outset. Generally, it is believed that DNA computers that use a brute-force search algorithm are limited to 60 to 70 variables (Lipton, 1995). Recently, a few new algorithms, such as the breadth-first search algorithm (Yoshida and Suyama, 2000), and random walking algorithm (Liu et al., 2005), have been proposed. With the breadth-first search algorithm, the capacity of a DNA computer can be theoretically increased to about 120 variables (Yoshida and Suyama, 2000).

In the present study, we developed a new space-efficient DNA computing algorithm based on a ligase chain reaction (LCR), which uses only four operations: ligating, amplifying, splitting and merging. Instead of generating the full DNA library at the beginning,
we start with an empty test tube, and generate partial solutions that only satisfy the clauses. The partial solutions are extended through ligation of new variable DNA: correct solutions are selectively amplified and false ones are pruned by LCR.

2. DNA Code Design

The SAT problem is an NP-hard computational problem that requires an exponential amount of time to solve with known sequential algorithms. Since all NP-

![Diagram of DNA computer]

Fig. 1. The principle of the proposed DNA computer. (A) Ligate $x^+_i$ with $x^+_j$ to form $x^+_i - x^+_j$, amplify the ligation product by LCR, and then ligate $x^+_k$ with $x^+_i - x^+_j$ to form $x^+_i - x^+_j - x^+_k$. Note that $s$ and $s'$ are respectively the sense and antisense strands of the linker sequence $S(5'-ACTTTCCC-3')$.

(B) Computation of the literal $x^+_j$: DNA strands that contain $x^+_j$ (or $x^+_0$) are (are not) amplified.

(C) Computation of the literals $\sim x^+_j$: DNA strands that contain $x^+_0$ (or $x^+_j$) are (are not) amplified.
complete problems can be encoded into SAT, the study of SAT problems has not only occupied a central role in theoretical computer science, but has also served as a benchmark for testing the performance of DNA computers. An n-variable m-clause 3-SAT problem can be written as,

\[ F = \bigwedge_{j=1}^{m} C_j = \bigwedge_{j=1}^{m} (\vee_{k=1}^{3} L_{jk}) \]

where \( C_j = (L_{j1} \vee L_{j2} \vee L_{j3}) \); \( \wedge \) is the logical AND operation; \( \vee \) is the logical OR operation; and the literals, \( L_{jk} \) (\( j = 1, 2, \ldots, m; \ k = 1, 2, 3 \)), are either a variable \( x_i \) or its negation \( \sim x_i \) (\( i = 1, 2, \ldots, n \)), where \( \sim \) is the logical NOT operation.

For simplicity, in this proof-of-principle study we target a simple example of the SAT problem with a 4-variable 8-clause 3-CNF Boolean formula,

\[ F = (x_0 \vee x_2 \vee x_1) \wedge (x_0 \vee x_1 \sim x_2) \wedge (x_0 \sim x_1 \vee x_2) \wedge (x_0 \sim x_1 \vee x_2) \]

Every possible solution can be represented by 4 binary variables written as \((x_0^i, x_1^i, x_2^i, x_3^i)\), where \( x_i \) (\( i = 0 \) to \( 3 \)) is the variable name; \( v \) is the value assignment of the variable, \( v = 1 \) represents True, and \( v = 0 \) represents False. There exists a unique correct solution \((x_0^1, x_1^1, x_2^1, x_3^1)\).

The data structure was designed in the form of double-strand DNA (dsDNA). Sixteen pairs of oligonucleotides were synthesized and phosphorylated (Table 1). The first eight pairs, \( x_i^+(+) \) and \( x_i^-(−) \), are respectively the sense and antisense strands of the variables. Since \( x_i^+(+) \) and \( x_i^-(−) \) are complementary with respect to each other, they can be annealed to form eight dsDNA molecules, \( x_{i} \), \( i = 0 \) to \( 3 \), \( v = 1 \) or \( 0 \). These molecules \( x_i \) are all 36 bp in length, include a 20-bp core sequence (underlined in Table 1), an 8-bp upstream 3’-extension (sticky end) and an 8-bp downstream linker (blunt end). The 20-bp core sequences represent the value assignments of the variables. The sequences of the sticky ends and the linkers are identical.

Another eight oligos, \( P_i^-(−) \), are used as primers for the ligase chain reaction, and their sequences are the same as that of \( x_i^-(−) \), \( i = 0 \) to \( 3 \), \( v = 1 \) or \( 0 \), except that they lack the sequences in the upstream 3’-extensions. The last eight oligos, \( Q_i^+(+) \), are all 20-bp long, and their sequences are the same as the core sequences of \( x_i^-(−) \) (\( i = 0 \) to \( 3 \), \( v = 1 \) or \( 0 \)), but without either the upstream 3’-extensions or the downstream linkers. \( Q_i^+(+) \) have two uses: to act as a primer for LCR and to be annealed with \( x_i^+(+) \) to form eight 28-bp headers, \( h_i \), whose core sequences are the same as that of \( x_i^+(+) \) (\( i = 0 \) to \( 3 \), \( v = 1 \) or \( 0 \)), but upstream of the core sequence is blunt-ended and the 3’-extensions are located downstream.

Since the downstream sticky end of \( x_i^+(+) \) and the upstream 3’-extensions of \( x_i^-(−) \) (\( i = 0 \) to \( 3 \), \( v = 1 \) or \( 0 \), respectively) are complementary with respect to each other, any \( h_i \) can be ligated with \( x_i^-(−) \) to form \( x_i^+(+) \) by Taq DNA ligase (Fig. 1A). The ligation product \( x_i^+(+) \) is \( x_i^+(+) \) can be amplified by a ligation chain reaction (LCR) with four primers, \( x_i^+(+) \), \( P_i^-(−) \), \( Q_i^+(+) \) and \( Q_i^+(−) \), by which the downstream sticky end will be regenerated in the product \( x_i^+(+) \), so that it can be ligated with \( h_i \) to form \( x_i^+(+) \) (Fig. 1A). Moreover, the ligation chain reaction has sufficient computational power to solve SAT problems: to compute a literal, correct strands are selectively amplified by a LCR reaction that contains only primers that satisfy the literal; false ones are pruned because primers that do not satisfy the literal are excluded (Fig. 1B and C).
3. DNA Computing Algorithm

In this computation, we will always start with one empty test tube. The set of DNA in the remaining test tubes corresponds to the graph $G_n$ (Fig. 2). The graph $G_n$ has $n$ nodes, $x_i$, $x_j$, $x_k$, $x_l$, . . . , where $i$, $j$, $k$, $l$ $\in$ [1, $n$]. Every node has two edges to every neighbor, and every edge comprises a logical gate. Every logic gate hold a fixed inner value of 0 or 1, which is conceptually new and different from conventional logic gates. A SAT formula is taken as a constraint condition to generate paths through the edges, the edges and the logic gates. All possible partial paths that start at the beginning and end at the latest node being treated are generated step by step for each literal and each clause. A path is called a true path and reserved if a literal in a clause is satisfied by the value of the corresponding node, otherwise the path is considered a false path and removed.

All true paths for all literals of a clause are merged to form a set of true paths for that clause; the true paths are then extended and/or sieved for the next clause. Therefore, all, and only, partial paths that satisfy all of the clauses that have been processed are generated. For example, given a SAT formula $(\neg x_i \lor \neg x_j) \land (x_i \lor \neg x_k)$, for the first clause, all paths through the first two nodes are generated, among which three true paths, $x_i^0 - x_j^0$, $x_i^1 - x_j^0$, and $x_i^0 - x_j^1$ are reserved, and the false path, $x_i^1 - x_j^1$, is removed. For the second clause, the true paths are extended to node $x_l$ to form six paths, among which four true paths, $x_i^0 - x_j^0 - x_l^0$, $x_i^1 - x_j^0 - x_l^0$, $x_i^0 - x_j^1 - x_l^0$, and $x_i^1 - x_j^0 - x_l^1$, are reserved, and the false paths, $x_i^0 - x_j^0 - x_l^1$ and $x_i^1 - x_j^1 - x_l^0$, are removed.

The pseudocode for solving a 3-SAT problem with $n$ variables and $m$ clauses is shown in Program 1. In the computing process, $t_j$ contains all of the sequences that satisfy clauses $C_1$ to $C_j$. Strings that do not satisfy $C_1$ to $C_j$ are not produced because the corresponding primers are absent from $t_{jk}$. After $m$ steps of such operations guided by the SAT formula, all correct strings that satisfy all of the clauses will be generated. The computation time is $O(3m + n)$ because split, ligation, LCR, and merge commands are executed $m$, $n$, $m$ and $m$ times, respectively.

*/ Program 1: Solve 3-SAT on a DNA computer */
Function LCR3SAT(F, $x_0$, $m$, $n$)
Begin */ Step 0: Begin with an empty test tube $t_0 */$
Split ($t_0$ $\rightarrow$ $t_{11}$, $t_{12}$, $t_{13}$)
Add ($t_{11}$, $t_{12}$, $t_{13}$, $x_1$)
For $j = 1$ to $m$ */ Step 1 to m: for each clause, $C_j = \lor \neg_{i=1}^3 (L_{ij}) */$
If $j > 1$ then Split ($t_{j1} \rightarrow$ $t_{j1}$, $t_{j2}$, $t_{j3}$)
In Parallel {
For $k = 1$ to $3$ */ for each literal of $C_j */$
If First appearance (F, $x_k$) then
For $l = 1$ to $3$
Ligate ($t_{jk}$, $x_k$)
Next l
End if
LCR ($t_{jk}$, L_{jk}) */ amplify $t_{jk}$ by LCR with primers that satisfy $L_{jk} */$
Next k
}
Merge ($t_{j1}$, $t_{j2}$, $t_{j3}$)
Next j
Return Detect ($t_{jm}$) /* Step m+1: detect the result by sequencing */
End Function

4. Implementation of the Algorithm

The biotechnological implementation of the proposed DNA algorithm is shown in Fig. 1. The commands are described in detail below:

(1) Ligate ($t_{jk}$, $x_i$), ligation of new variables, was performed in a volume of 40 µL containing 1 µmol/L header or LCR product, 1 µmol/L $x_i^0$, 1X Taq DNA ligase buffer and 80 U Taq DNA ligase (NEB). This mixture was heated to 95 °C for 1 min, gradually cooled to 55 °C, and then incubated at 55 °C for 30 min.

(2) LCR ($t_{jk}$, L_{jk}), selective amplification of the ligation products in $t_{jk}$ according to L_{jk}, is performed by a LCR reaction that contains only primers that satisfy the literal L_{jk}. For example, amplification of $x_i^0 - x_j^0 - x_l^0$ according to $\neg x_i$ was performed in a total volume of 50 µL, using 10 ng of ligation product $x_i^0 - x_j^0 - x_l^0$, 100 nmol/L of each primer $x_i^0$, $x_j^0$, $x_l^0$, $P_i^0$, $P_j^0$, and $Q_i^0$, and 20 U Taq DNA ligase in 1X Buffer (NEB). Amplification was carried out on a Biometre T-gradient thermal controller as follows: pre-denaturation at 94 °C for 1 min, followed by 25 cycles of denaturing at 94 °C for 20 s, annealing and ligation at 60 °C for 2 min, and a final ligation at 60 °C for 10 min.
5. Results of the Lab Experiment

The DNA samples generated in each step were detected by electrophoresis on 15% polyacrylamide gel for 0.5 h at 240 V, results are shown in Fig. 3. To compute the first clause, DNA fragments $hx_0^v$ (28 bp) were added into three empty test tubes, $t_{11}$, $t_{12}$ and $t_{13}$, ligated with $x_v^2$ (36 bp) to form $x_v^0 - x_v^2$ (64 bp), amplified by LCR to regenerate the sticky end, and then ligated with $x_v^1$ to produce $x_v^0 - x_v^2 - x_v^1$ (92 bp). DNA strands in $t_{11}$, $t_{12}$ and $t_{13}$ were selectively amplified by LCR, which removed incorrect solutions, and merged into $t_1$. DNA in $t_1$ ($x_v^0 - x_v^2 - x_v^1$) was split into $t_{21}$, $t_{22}$ and $t_{23}$, and further computation was carried out using program 1 in a series of test tubes $t_j$ according to the corresponding clause $C_j$, $j = 2, 3 \ldots, 8$. Note that to compute the sixth clause ($\neg x_0^0 \lor \neg x_3^0 \lor \neg x_2^0$), since this is the first appearance of $x_3$, DNA in $t_{61}$, $t_{62}$ and $t_{63}$ were ligated with $x_3^v$ to form $x_0^v - x_2^v - x_3^v - x_0^3$ (120 bp), then selectively amplified by LCR and merged into $t_6$. The 64 bp bands shown in lane $t_1$ through $t_8$ are incomplete solutions, such as $x_i^v - x_j^v$, formed in the LCR reactions; these incomplete solutions do not cause serious problems, since they can be removed by size separation or recognized easily by sequencing of the final answer DNA.

After eight steps of operations guided by the SAT formula, the answer DNA was finally produced in test tube $t_8$. The answer DNA in each step was detected by molecular cloning and sequencing. The answer DNA in $t_1$ through $t_8$ was re-amplified by LCR to make both ends blunt, and the LCR product was separated by electrophoresis on 15% polyacrylamide gel for 0.5 h at 240 V. After EB staining, the desired DNA band was sliced under UV-light, purified and ligated into pUC19 plasmid that had been digested with EcoRV. The plasmid containing the answer DNA was transferred into E. coli JM109 and cloned, and then for each sample 50 colonies were selected and propagated in liquid LB medium containing 50 μg/mL ampicillin. The plasmid DNA was then extracted and sequenced. The sequences were decoded according to Table 1 and classified. The results are consistent with the expectation (Table 2). The sequencing results of the last test tube ($t_8$) are identical (Fig. 4), and from these the answer was decoded into $x_1^0 - x_2^0 - x_0^1 - x_0^3$. Thus, this simple example of a 3-SAT problem was solved using this DNA computing algorithm.

6. Evaluation of the Space Complexity of the DNA Algorithm

For a given $m$-clause SAT formula, $F$, in the computing process, when $j$ grows from 1 to $m$, the number of different DNA strands in $t_j$ equals the number of true assignments ($N_j$) for the first $j$ clauses. The space complexity of this algorithm is the maximum number of DNA
strands produced in test tubes $t_j$, $\max \{N_j| j = 1, \ldots, m\}$, which is always smaller than the full-solution space ($2^n$).

To investigate the space complexity of this algorithm, randomly generated 3-CNF SAT problems were solved using a home-made simulation program running on an HP Proliant workstation and an 80-node PC cluster. To generate sample formulas, we program that gives a range for the number of variables, $n_1$ to $n_2$, and a range for the clause/variable ratios, $r_1$ to $r_2$, and constructs formulas of $n$ variables and $m$ clauses, with $n \in [n_1, n_2]$, $m/n \in [r_1, r_2]$. When picking up a clause, three literals were repeatedly selected independently with equal probability, while the clause was kept free from complementary literals and identical literals.

In both conventional and molecular computing studies on SAT problems, we are mainly concerned with hard SAT problems. It is well known that random $k$-SAT undergoes a phase transition in computational complexity for $k > 2$. This transition occurs for specific values of $m/n$, the ratio of the number of clauses ($m$) to the number of variables ($n$). For low values of $m/n$ the problem is easy to solve and there is a large number of possible valuations for the variables that solve the problem; in this regime we say that the problem is under-constrained. For large values of $m/n$, the problem is over-constrained and it is easy to discover early in the search process that no solution to the problem exists. However, around the critical value $m/n$, the problem becomes computationally hard and sometimes there is only a single valuation, out of the $2^n$ possible combinations, that solves the formula. In the case of random 3-SAT, it is well known that the clause/variable ratio of the hardest instances of 3CNF-SAT is around 4.25 (Selman et al., 1996). Therefore, we generated 50,000 instances of random 3CNF-SAT formulas, with number of variables $n \in [5, 50]$ (more than 1000 for each $n$) and clause/variable ratio $m/n \in [1, 50]$, and then investigated how the number of partial assignments changes as the algorithm runs.

Selective LCR amplification helps to decrease the number of partial assignments, and the more often a

<table>
<thead>
<tr>
<th>Test tube no.</th>
<th>Clause treated</th>
<th>Sequence decoded</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$C_1 (x_0 \lor x_1 \lor x_2)$</td>
<td>$x_0^0 - x_0^0 - x_1^1 - x_2^0$, $x_0^0 - x_2^0 - x_1^1$, $x_0^0 - x_1^1 - x_1^1$, $x_0^0 - x_1^1 - x_2^0$, $x_0^0 - x_1^0 - x_2^0$, $x_0^0 - x_2^0 - x_1^0$, $x_0^0 - x_2^0 - x_1^0$, $x_0^0 - x_1^1 - x_2^0$</td>
</tr>
<tr>
<td>2</td>
<td>$C_2 (x_0 \lor x_1 \lor \neg x_2)$</td>
<td>$x_0^0 - x_0^0 - x_1^0 - x_2^0$, $x_0^0 - x_2^0 - x_1^0$, $x_0^0 - x_1^0 - x_2^0$, $x_0^0 - x_1^0 - x_1^0$, $x_0^0 - x_2^0 - x_1^0$, $x_0^0 - x_2^0 - x_1^0$, $x_0^0 - x_2^0 - x_1^0$, $x_0^0 - x_1^0 - x_2^0$</td>
</tr>
<tr>
<td>3</td>
<td>$C_3 (x_0 \lor x_1 \lor x_2)$</td>
<td>$x_0^0 - x_0^0 - x_1^0 - x_2^0$, $x_0^0 - x_2^0 - x_1^0$, $x_0^0 - x_1^0 - x_2^0$, $x_0^0 - x_1^0 - x_1^0$, $x_0^0 - x_2^0 - x_1^0$, $x_0^0 - x_2^0 - x_1^0$, $x_0^0 - x_2^0 - x_1^0$, $x_0^0 - x_1^0 - x_2^0$</td>
</tr>
<tr>
<td>4</td>
<td>$C_4 (x_0 \lor x_1 \lor \neg x_2)$</td>
<td>$x_0^0 - x_0^0 - x_1^0 - x_2^0$, $x_0^0 - x_2^0 - x_1^0$, $x_0^0 - x_1^0 - x_2^0$, $x_0^0 - x_1^0 - x_1^0$, $x_0^0 - x_2^0 - x_1^0$, $x_0^0 - x_2^0 - x_1^0$, $x_0^0 - x_2^0 - x_1^0$, $x_0^0 - x_1^0 - x_2^0$</td>
</tr>
<tr>
<td>5</td>
<td>$C_5 (x_0 \lor x_1 \lor x_2)$</td>
<td>$x_0^0 - x_0^0 - x_1^0 - x_2^0$, $x_0^0 - x_2^0 - x_1^0$, $x_0^0 - x_1^0 - x_2^0$, $x_0^0 - x_1^0 - x_1^0$, $x_0^0 - x_2^0 - x_1^0$, $x_0^0 - x_2^0 - x_1^0$, $x_0^0 - x_2^0 - x_1^0$, $x_0^0 - x_1^0 - x_2^0$</td>
</tr>
<tr>
<td>6</td>
<td>$C_6 (x_0 \lor x_1 \lor x_2)$</td>
<td>$x_0^0 - x_0^0 - x_1^0 - x_2^0$, $x_0^0 - x_2^0 - x_1^0$, $x_0^0 - x_1^0 - x_2^0$, $x_0^0 - x_1^0 - x_1^0$, $x_0^0 - x_2^0 - x_1^0$, $x_0^0 - x_2^0 - x_1^0$, $x_0^0 - x_2^0 - x_1^0$, $x_0^0 - x_1^0 - x_2^0$</td>
</tr>
<tr>
<td>7</td>
<td>$C_7 (x_0 \lor \neg x_1 \lor x_2)$</td>
<td>$x_0^0 - x_0^0 - x_1^0 - x_2^0$, $x_0^0 - x_2^0 - x_1^0$, $x_0^0 - x_1^0 - x_2^0$, $x_0^0 - x_1^0 - x_1^0$, $x_0^0 - x_2^0 - x_1^0$, $x_0^0 - x_2^0 - x_1^0$, $x_0^0 - x_2^0 - x_1^0$, $x_0^0 - x_1^0 - x_2^0$</td>
</tr>
<tr>
<td>8</td>
<td>$C_8 (x_0 \lor \neg x_1 \lor x_2)$</td>
<td>$x_0^0 - x_0^0 - x_1^0 - x_2^0$, $x_0^0 - x_2^0 - x_1^0$, $x_0^0 - x_1^0 - x_2^0$, $x_0^0 - x_1^0 - x_1^0$, $x_0^0 - x_2^0 - x_1^0$, $x_0^0 - x_2^0 - x_1^0$, $x_0^0 - x_2^0 - x_1^0$, $x_0^0 - x_1^0 - x_2^0$</td>
</tr>
</tbody>
</table>
variable appears, the more LCR selections it brings. Therefore, we adopted a selection-first strategy: the clauses are scored and sorted to make the selections as soon as possible after ligating a new variable. Once a 3-SAT formula is generated, say \( F = C_1 \land C_2 \land \ldots \land C_m \), the clauses, \( C_j \), are scored by,

\[
W_j = \frac{1}{n} \log(q_i);
\]

\[
W_j = \sum_{i=1}^{m} \log(1 + 1)/n;
\]

\[
W_j = W_{j1}/n + W_{j2}/m; \quad j = 1 \ldots , m,
\]

where \( i \) and \( q_i \) are, respectively, the indices and the occurrence number of the variable \( x_i \) for the literal \( L_{jk} \). The clauses are then sorted in descending order by \( W_j \), by which \( F \) was converted into an equivalent form, \( F' = C'_1 \land C'_2 \land \ldots \land C'_m \); where \( C'_j \in \{ C_1 , C_2 , \ldots , C_m \} \), \( W'_1 > W'_2 > \ldots > W'_m \). The order of the clauses is determined mainly by \( W_{j1} \) and fine-tuned by \( W_{j2} \). Next, \( F' \) was solved with the simulation program for this algorithm running on an electronic computer, which gave the maximum number of partial assignments (\( \lambda \)) and the exponent ratio, \( \rho = (\log(\lambda))/n \). The average and maximum ratio \( \rho \) for random 3-SAT problems are shown in Fig. 5. The number of assignments in the initial pool generated by Lipton’s brute-force algorithm is \( 2^n \), so the exponent ratio for a brute-force algorithm is a constant, \( \rho_{\text{Lipton}} = 1.0 \). The observed ratio \( \rho \) for this algorithm decreases almost linearly with an increase in \( n \) and the \( m/n \) ratio. When \( n=50 \), the overall average and maximum number of DNA strands required is, respectively, \( 2^{0.4198n} \) and \( 2^{0.488n} \). If this relation \( 2^{0.488n} \) holds true or decreases further in cases of 3-SAT problems with more variables, this algorithm should make it possible to solve large and hard 3-SAT problems with a much smaller amount of DNA than the conventional brute-force method. Additionally, as Fig. 6 indicates, the worst case situation \( 2^{0.488n} \) is rare; in most cases the ratio is less than \( 2^{0.446n} \). Thus, based on the analysis in Section 3 and Section 6, we propose the hypothesis that most large and hard SAT problems can be solved on a DNA computer with time complexity of at most \( O (3m + n) \) and space complexity of less than \( 2^{0.488n} \).

### 7. Discussion

Even though the sample SAT problem solved here is very small, the proposed DNA computing algorithm has several advantages over the conventional brute-force search algorithm. First, it eliminates the need to construct a full-solution DNA library. The initial test tube \( (t_0) \) is empty instead of containing the full-solution data pool, and the other test tubes \( t_j (j = 1 \text{ to } m) \) contain only strings that satisfy clauses \( C_1 \) to \( C_j \), which greatly reduces the number of DNA strands required in the DNA computation and makes it possible to extend this approach to solve large combinatorial problems.

Second, the special characteristics of the proposed graph that was used to construct the paths to solve the SAT problem not only help to reduce the number of DNA strands required, but also bring some new concepts into the graph theory of combinatorial problems. The proposed graph differs from Lipton’s mainly in two aspects. First, every edge is embedded with a logic gate that holds a value assignment for the corresponding node and checks whether a path is allowed or not, and paths are generated step by step, restricted by the clauses of the SAT formula. Therefore, only true paths that satisfy the clauses being treated are reserved. In contrast, in Lipton’s graph, all \( 2^n \) possible paths are generated at the beginning of computation. Second, the nodes are sequenced accord-
ing to the order of the first appearance of the variables in the SAT formula instead of by their indices. Compared to previous algorithms (Adleman, 1994; Lipton, 1995; Yoshida and Suyama, 2000; Liu et al., 2005; Ouyang et al., 1997; Sakamoto et al., 2000; Braich et al., 2002; Liu et al., 2000; Su and Smith, 2004; Faulhammer et al., 2000; Ogihara and Ray, 2000) in which the variables are commonly connected in the increasing order of their indices, this feature of the present algorithm makes it much easier to handle and possible to score and sort clauses, which makes the searching space smaller.

Third, this DNA computing algorithm is highly error-tolerant. In our experimental implementation of this algorithm, only one enzyme, Taq DNA ligase, was used to carry out all of the enzymatic operations. Taq DNA ligase is a highly accurate DNA ligase that has only nick-repairing activity. It can only ligate perfectly matched sticky ends, which means that a one-base-pair mismatch in the sticky ends is enough to prevent them from ligation DNA molecules. The intrinsic highly accurate sequence-specific DNA ligation ability of Taq DNA ligase makes it probably one of the best tools for use in DNA computing. Other enzymatic DNA operations, such as PCR amplification and restriction digestion, have been used successfully to solve the max clique problem (Ouyang et al., 1997). Those authors pointed out that major errors arise from two sources. The first is the production of single-stranded DNA (ssDNA) during PCR, which cannot be cut by restriction enzymes. The second is the incomplete cutting of double-strand DNA (dsDNA) by restriction enzymes, which also leads to incorrect answers (Ouyang et al., 1997).

Previously, we solved a SAT problem using PCR to amplify the ligation product, and restriction endonuclease digestion to prune incorrect DNA strands (unpublished). However, these methods are obviously impractical for large problems, since it will become increasingly difficult to handle more and more different restriction enzymes, which works often in different buffer solutions. In the present study, all of the enzymatic operations, including ligating, amplifying and pruning, were carried out with the same enzyme that works in the same buffer. This makes the whole computing process simple, fully automatable, and practical for large problems.

Generally, PCR-based algorithms are error-prone, since the key enzyme used in the PCR reaction, Taq DNA polymerase, produces not only desired DNA strands but also sometimes unwanted sequences, since it is possible that the primers may anneal non-specifically to the wrong sequences (mismatching primers). More serious errors, such as point mutations in the restriction sites, may sometimes arise when PCR is used to amplify the solution DNA, which is often required to contain many restriction sites (Ouyang et al., 1997). In PCR, primers that are annealed to the template are elongated nucleotide by nucleotide, which is the main source of point mutation. In contrast, in LCR, they are extended block by block, since all DNA sequence blocks are synthesized before the beginning of computation, and during computation we only have to ligate these pre-synthesized sequence blocks properly.

In this LCR-based algorithm, all of the above-mentioned sources of errors, such as point mutations, mismatching primers, incomplete cutting of ssDNA and dsDNA, are eliminated. Thanks to the high fidelity of LCR, this procedure gives an exponential amplifier with a larger exponent for correct strands than for incorrect ones. Therefore, by repeating the LCR process, we should reduce the amount of noise arising from the remaining false strands in the template DNA. Therefore, this algorithm is inherently more reliable than conventional PCR-based algorithms.

As noted by Adleman (1994), the major advantages of DNA computing lie in its high information-storage capacity and parallel-computation power. The present algorithm takes advantage of these features of DNA computing without generating an initial data pool that contains every possible solution. Compared with conventional brute-force algorithms, this algorithm is more space-efficient and error-tolerant. Though the number of DNA strands required still increase exponentially with the size of the problem, the exponent ratio is less than half of that for brute-force algorithms. We believe that it can be scaled-up to solve large and hard SAT problems. The maximum number of variables that it can deal with depends on how many cycles of ligation and LCR amplification can be performed to extend the DNA strands without any serious error. In the present study, we performed three steps of extension and obtained a 4-bit DNA solution. This process can theoretically proceed for as many steps as desired, while the actual number of steps should be determined in practice by further experiments.

In conclusion, we believe that the algorithm described here represents an important contribution to the development of DNA computing models, and there is great potential for the further exploration of DNA computers. This algorithm starts with an empty test tube, since an empty start actually means all of the possibilities are included. With continual improvements in both DNA computing algorithms and molecular biotechnologies, it may soon be possible to develop powerful DNA computers that can be useful in certain aspects, such as
in an *in vivo* biochemical/genetic controlling system. Unfortunately, at present, only the data are encoded into DNA sequences; the programs used for data processing are still controlled by humans or with the aid of electrical-mechanical systems. One of the most interesting open questions is how to create a truly molecular computer (TMC), which means that not only the data, but also the programs themselves are encoded into DNA sequences, and execution of the program is not controlled by a human but rather by an automatic biological system.

**Acknowledgements**

This research was supported by the National Science Foundation of China through Grant 30500379.

**References**


