Analyzing Complexity For NP-Complete Problem Through DNA Computing Algorithm

Shalini Rajawat [1], Dr Vijay Singh Rathore [2],Naveen Hemrajani [1],Ekta Menghani[3]

GJCST Classification (FOR) F.1.3. F.2.1. F.2.2

Abstract*- Adleman and Lipton adopted a brute-force search strategy to solve NP-complete problems by DNA computing i.e., a DNA data pool containing the full solution space must first be constructed in the initial test tube (*t*0), and then correct answers are extracted and/or false ones are eliminated from the data pool step by step. Thus, the number of distinct DNA strands contained in the initial test tube (*t*0) grows exponentially with the size of the problem. The number of DNA strands required for large problems eventually swamps the DNA data storage, which makes molecular computation impractical from the outset. Lipton's brute-force search DNA algorithm is limited to about 60 to 70 variables and thus it is believed that DNA computers that use a brute-force search algorithm can not exceed the performance of electronic computers. Since then, studies on DNA computing have focused on reducing the size of the data pool. A few new algorithms, such as the breadth-first search algorithm , Genetic algorithm , random walking algorithm , have been proposed and tested. With the breadth-first search algorithm, the capacity of a DNA computer can be theoretically increased to about 120 variables, but even so, DNA computers are still not capable of competing with electronic computers. Previously, we solved the SAT problem using a DNA computing algorithm based on ligase chain reaction. In the present study, we solve the SAT problem with the same DNA computing strategy using alternative biotechnical operations. Here we report some new results on the universality and space complexity of this DNA computing algorithm.

Keywords-DNA computing, NP-Complete, space complexity, time complexity

I. DNA COMPUTING ALGORITHM

Without becoming too specific, we can assume that none of the clauses of *F* has both the positive form and negative form of the same variable and that *F* does not have two or more clauses consisting of the same three literals. The program for solving a 3-SAT problem with *n* variables and *m* clauses is shown in *Program. 1*. In the computing process, *tj* contains all of the sequences that satisfy clauses *C*1 to *Cj*. Strings that do not satisfy *C*1 to *Cj* can not be produced because the corresponding variable DNA is absent in *tjk*, or can not be amplified by PCR because they are broken by a restriction enzyme in *tjk*. 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* (9*m*+3*n*) because *Split*, *U-ligate*, *Cut*, *Amplify*, *Merge* and *Detect* commands are executed at most *m*, 3*n*, 3*m*, 3*m*, *m* and *m* times in the program, respectively.Therefore, the NP-complete problem can be solved in an amount of time that is proportional to the size of the problem.

About[1]-Department of Computer Science & Engineering, Suresh GyanVihar University, Jaipur, INDIA (rajawatshalini@yahoo.co.in; naven_h@yahoo.com
About[2]-Department of Computer Science & Engineering, Karni College, Jaipur INDIA. (vijaydiamaond@gmail.com)
About[3]-Department of Biotechnology, Mahatma Gandhi Institute of Applied Sciences, Jaipur, INDIA (ekta.menghani@rediffmail.com)

II. IMPLEMENTATION OF THE ALGORITHM

Biotechnological implementation of the DNA algorithm is shown in Fig. (**1**). The commands are described in detail below:

(1) *PCR amplification of* *x*0 *v-xi v* was performed in a total volume of 50μL, using 100 nmol/L of each primer *P*0 and *Pi*, 10ng of ligation product *x*0 *v-xi v*, 200 mmol/L of each of the 4 dNTP, and 2.5U *Taq* DNA polymerase in 1X PCR Buffer supplemented with MgCl2
at a final concentration of 1.5 mM (all from Promega). Amplification was carried out on a Biometre T1 thermal controller as follows:
predenaturing at 94°C for 1 min, followed by 20 cycles of denaturing at 94°C for 20s, annealing at 62°C for 20s, and extension at 72°C for 20s, and a final extension at 72°C for 1 min.

(2) *U-ligation of variables xj v to x*0 *v-xi v* was performed in a volume of 20μL, containing 100ng PCR product *x*0 *v-xi v*, 1X PCR buffer and 2U *USER* enzyme (*NEB*). This mixture was incubated at 37□for 30 min to cut the uracil base. Next, 1μmol *xj v*, 1X *Taq* DNA ligase buffer and 80U *Taq* DNA ligase (*NEB*) were added,and the mixture was heated to 95 c for 5 min, gradually cooled to 55 c, and incubated at 55 c for 30 min to ligate *xj v* and *x*0 *v-xi v*.

(3) *Restriction cutting of* *x*0 *v-xi v*... was performed in a volume of 20μL containing 100ng PCR product *x*0 *v-xi v*..., 1X restriction buffer and 20U restriction enzyme (*NEB*) selected according to *Table* 1, and this mixture was incubated at the temperature recommended by the manufacturer for 60 min to cut strings containing *xi v*.

```
/* Program 1: Solve 3-SAT on a DNA computer */
Function DNA3SAT (F, xi, m, n)
Begin
Empty (t0) /* Begin with an empty test tube t0 */
/* Step 1 to m: for each clause of F, Cj =v3 k = (Ljk)*/
For j = 1 to m
Split (tj-1, tj1, tj2, tj3) /* Split test tube tj-1 equally to tj1, tj2, tj3 */
```

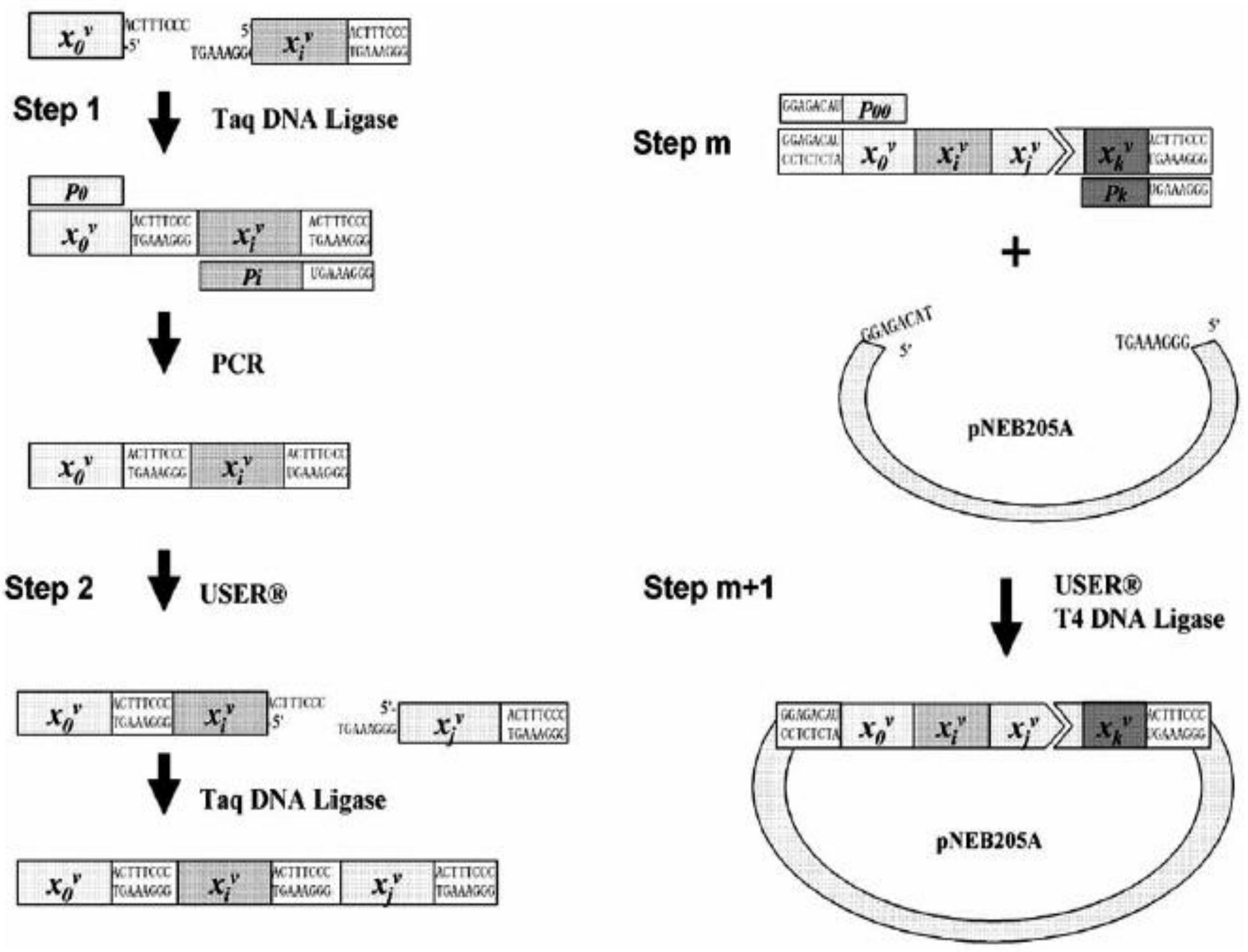

Fig. (1). Biotechnological operations in each step of the DNA computing process:

Step 1: *Ligating* of *x*0 *v* and *xi v* and PCR amplification of the product *x*0 *v-xi v*;
Step 2: *U-ligating xj v: USER-cutting* the ligation product *x*0 *v-xi v* to regenerate the sticky end and *ligating* of *xj v* to get *x*0 *v-xi v-xj v*;
Step *m*: At the end of computation, PCR is used to amplify the final answer DNA *x*0
v-xi v-xj v-…*-xk v* with primer *P*00 and *Pk*;
Step *m*+1: Cloning of PCR product of the final answer DNA into the pNEB205A vector.

```
For k = 1 to 3 /* for each literal of Cj */
i = index (Ljk)
If First_occurrence (F, xi) then /* If Ljk (xi or ~xi) is the
first occurrence
of xi in F */
For l = 1 to 3
U-ligate (tjl, xi
0); U-ligate (tjl, xi
1)
Next l


End If
Restriction_cutting (tjk, NOT Ljk) /* Cut NOT Ljk with
restriction
enzyme */
PCR_ amplification (tj1, tj2, tj3, P0, Pi)
/* PCR Amplify DNA in tj1, tj2 and tj3 with primer P0 and
Pi */
Next k
Merge (tj, tj1, tj2, tj3) /* Merge test tube tj1, tj2, tj3 to tj */
Next j
/*Step m+1: detect the result by sequencing*/
Return Detect (tm)
End Function
```

III. Results of the Lab Experiment

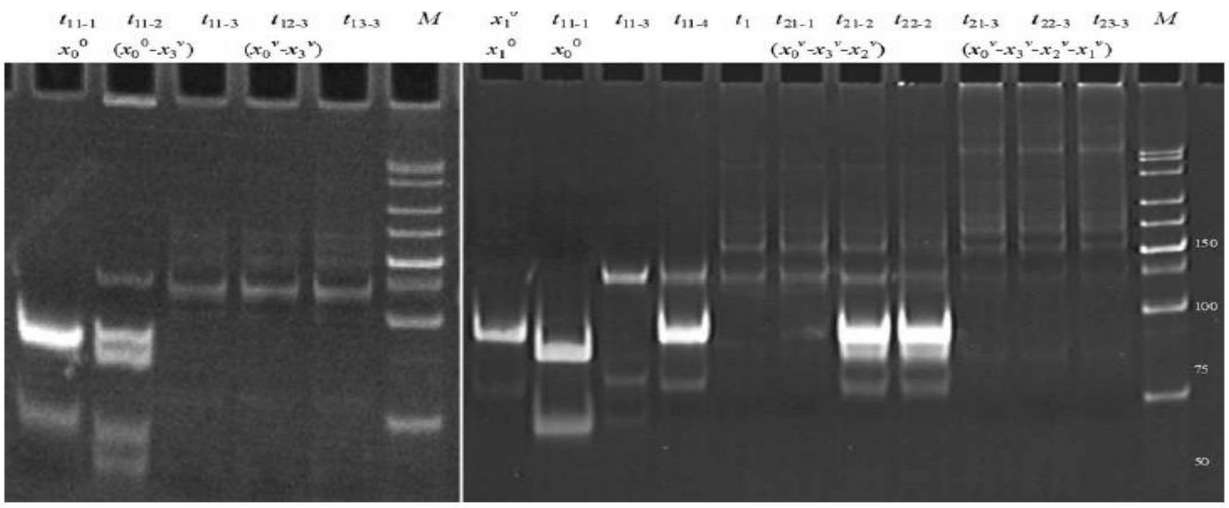
$t_{11\text{-}1}$ x_0^0
$t_{11\text{-}2}$ $(x_0^0\text{-}x_3^v)$
$t_{11\text{-}3}$
$t_{12\text{-}3}$ $(x_0^v\text{-}x_3^v)$
$t_{13\text{-}3}$
M
x_1^v x_1^0
$t_{11\text{-}1}$ x_0^0
$t_{11\text{-}3}$
$t_{11\text{-}4}$
t_1
$t_{21\text{-}1}$
$t_{21\text{-}2}$ $(x_0^v\text{-}x_3^v\text{-}x_2^v)$
$t_{22\text{-}2}$
$t_{21\text{-}3}$
$t_{22\text{-}3}$ $(x_0^v\text{-}x_3^v\text{-}x_2^v\text{-}x_1^v)$
$t_{23\text{-}3}$
M
150
100
75
50

Fig. (2). Results of the DNA computing process (Steps **0** to **2**).

M is a 25-bp DNA ladder (MBI) DNA bands show *x*0 0 in test tube *t*11-1, *ligating* product in test tube *t*11-2 (*x*0 0- *x*3 *v*) and PCR products in test tubes *t*11-3, *t*12-3, *t*13-3, *t*11-3, *t*11-4, *t*1, *t*21-1, *t*21-2, *t*22-2, *t*21-3, *t*22-3 and *t*23-3, respectively.

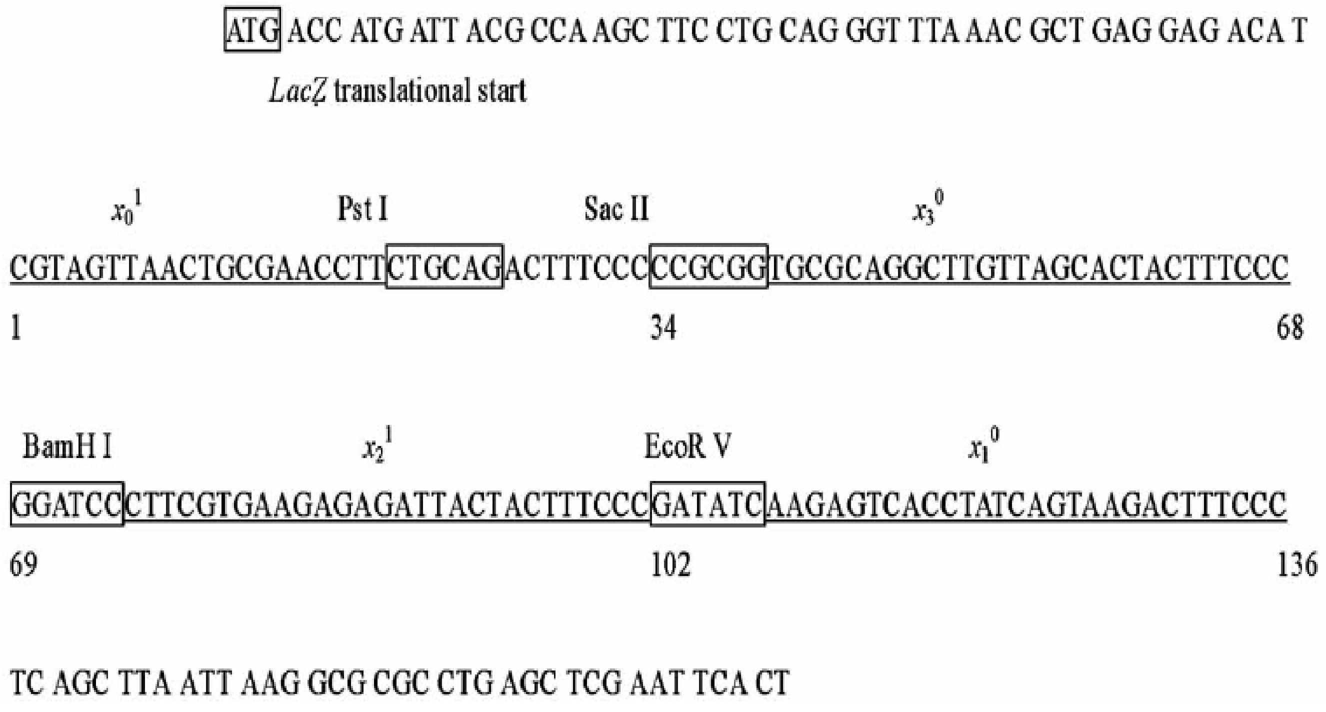
ATG ACC ATG ATT ACG CCA AGC TTC CTG CAG GGT TTA AAC GCT GAG GAG ACA T
LacZ translational start
x_0^1
Pst I
Sac II
x_3^0
CGTAGTTAACTGCGAACCTTCTGCAGACTTTCCCCCGCGGTGCGCAGGCTTGTTAGCACTACTTTCCC
1
34
68
BamH I
x_2^1
EcoR V
x_1^0
GGATCCCTTCGTGAAGAGAGATTACTACTTTCCCGATATCAAGAGTCACCTATCAGTAAGACTTTCCC
69
102
136
TC AGC TTA ATT AAG GCG CGC CTG AGC TCG AAT TCA CT

Fig. (3). Result of DNA cloning and sequencing of the final answer DNA. The variable names and values are marked, the binding positions of primers *P*0 to *P*3 are underlined, the restriction sites are boxed and marked, *Pst I-Sac II-BamH I-EcoR V*, and the unique answer is *x*0 1-*x*3

IV. Evaluation of the Space Complexity of the DNA Algorithm

For a given *m*-clause random SAT formula, *F*, the first *j* clauses is a SAT formula with *j* clauses, say *Fj*. In the computing process, when *j* grows from 1 to *m*, the number of different DNA strands in *tj*, say *Nj*, equals to the number of true assignment of *Fj*. The space complexity of this algorithm is the maximum number of DNA strands produced in test tubes *tj*, or the maximum number of partial assignments of Fj, say $max\{Nj, j = 1,...,m\}$, which is always smaller than the full solution space ($2n$).

We have examined the performance of our algorithm by computer simulation. We implemented on a HP Proliant workstation running a program that simulates our SAT solving algorithm on a family of random generated 3-CNF SAT formulas. In order to generate sample formulas, we wrote a program that give a range for the number of variables, $n1$ to $n2$, and a range for the clause/variable ratios, $r1$ to $r2$, constructs formulas of n variables and m clauses, where $n€$ [$n1$, $n2$], $m/n€$ [$r1$**,** $r2$]. When picking up a clause, three literals are repeatedly selected independently with equal probability, while keeping the clause free from complementary literals and identical literals. In both conventional and molecular computing studies on SAT problem, what we interested in are *hard* SAT problems. The clause/variable ratio of the hardest instances of 3CNF-SAT is around 4.3 [1, 16]. There are several algorithms that very efficiently solve the SAT problem if the clause/variable ratio is even slightly off from the critical point near 4.3. In order to test our SAT solving algorithm on both *easy* and *hard* problems, we generated at random fifty thousand instances of 3CNF-SAT problems with number of variables $n€(5, 50)$ (more than one thousand for each n) and clause/variable ratio $m/n€$ (1, 50), and then investigated how the number of partial assignments changes while the algorithm runs. Because *cutting* operation helps decreasing of the partial assignments, and the more frequent a variable occur, the more *cutting* operation it brings to the computing. So we adopted a *cutting-first* strategy, the clauses are scored and sorted so that the *cutting* operations would happen as soon as a new variable is ligated to the solution.

Once a 3-SAT formula is generated, say $F=C1^\wedge C2^\wedge\ldots Cm$, the clauses, Cj, are scored by, $Wj = \pi 3\ k=1 \log (qi), j = 1,\ldots, m$, Where qi is the occurrence number in F of the variable (xi) for theliteral Ljk. Then the clauses are sorted descendingly according to Wj, F was then transformed into an equivalent form, $F' = C1'^\wedge C2'^\wedge\ldots Cm'$; where $Cj€$ [$C1$**,** $C2$, …, Cm], $W1' >W2'>...>Wm'$.Then we solve F' using the sequential version of our algorithm running on electronic computer and computes the maximum number of partial assignments (γ) that are required, outputs the exponent ratio $p= (\log 2\gamma)/n$. The average and maximum ratio γ for the maximum number of necessary partial assignments in solving random 3- SAT problems is shown in Fig. (**4**). The number of assignments in the initial pool generated by the brute force algorithm is $2n$, so theratio γ for Lipton's brute force algorithm is a constant, □$Lipton$=1.0. The observed ratio γ for our algorithm decreases almost linearly with the increasing of n and m/n ratio. When $n = 50$, the overall average and maximum for the maximum number of DNA strands required is 20**.**4198n and 20**.**48n, respectively. If this relation 20**.**48n holds true or decreases further in 3-SAT instances with more variables,our algorithm will make solution of large and hard 3-SAT problemwith much smaller amount of DNA than the conventional bruteforce method. The observed average and maximum exponent ratio for this algorithm decreases logarithmically with the increasing of n. The regression equation of the maximum ratio to the number of variables n is,= 1.2902 -0.1788 ln (n) When n is set to be 100 and 200, the predicted maximum number of DNA strands required is 1.13E+14 and 4.39E+20, *i.e.*, the amount of DNA strands required are respectively within several nanomole and micromole. These requirements are surely possible with current biotechniques. If this relation holds true, this algorithm will make the solution of large 3-SAT problem possible with much smaller amount of DNA than the conventional brute-force method. Thus, based on the analysis in *section 3* and *section 6*, we proposed conjecture: For the class of SAT problems generated by our program for random generated 3-CNF SAT formulas can be solved on a DNA computer with time complexity O (9m) and space complexity2 [1.2902-0.1788 ln (n)]n.

V. Discussion

Even though the sample SAT problem solved here is very small, the proposed DNA computing algorithm has several advantages. Firstly, it eliminates the need to construct a full-solution 0-$x2$ 1-$x1$ 0. DNA library. The first test tube ($t0$) is empty instead of containing the full-solution data pool, and the other test tubes tj (j=1 to m) contain only strings that satisfy clauses $C1$ to Cj, which greatly reduces the number of DNA strands needed in the DNA computation, and makes it possible to extend this approach to solve large SAT problems, and possibly to other large NP-problems[2]. The maximum number of variables it can deal with depends mainly on how many cycles of *U-ligation* and *amplification* can be performed to extend the DNA strands without any serious error. In the present study, we performed 3 steps of extension and obtained a 4-word DNA solution. Although the process can theoretically proceed for as many steps as desired, the actual number of steps should be determined by further experiment in practice; Secondly, our DNA computation algorithm is error-tolerant. In this algorithm, we adopted *U-ligating*, *PCR amplifying* and *restriction cutting* as basic operations. As far as we know, ligase and restriction enzyme are both the most precise DNA operation enzymes available[3]. A one-base-pair mismatch in the restriction sites or in the sticky ends is enough to prevent them from cutting or ligating DNA molecules. The intrinsic highly accurate DNA sequence-recognition ability of DNA ligase and restriction enzyme makes them the most suitable tools for use in DNA computing. The same operations have been used successfully to solve the max clique problem [7], and those authors pointed out that the major errors in this computation arise from two sources. The first is the production of single stranded DNA (ssDNA) during PCR. This ssDNA cannot be cut by restriction enzymes. The second source of errors is incomplete cutting of double-strand DNA (dsDNA) by restriction enzymes, which also leads to incorrect answers. We used the selected restriction enzymes in 10-fold over digestion and found that they work well enough for our purpose in one cycle of digestion-PCR. Thanks to the combination of restriction digestion and PCR, this procedure gives an exponential amplifier with a larger exponent for uncut strands than for cut strands. Repeating the digestion-PCR process should therefore reduce the amount of noise arising from incomplete digestion [4-5]. In addition, the *U-ligating* operation we used to extend DNA strands not only helps to resist errors, but also increases the practical capacity of the DNA computer, since it is not only fast, easy and effective, but also prevents unwanted DNA strands from being generated by avoiding mistaken-ligation. Thirdly, the variables in the solution DNA are linked in the order of their position in the SAT formula instead of their indices. Compared with previous algorithms in which the variables are usually connected in the order of their indices [4-8], this feature of our algorithm makes it much easier to handle and possible to implement DNA computing without generating the full solution pool. In our algorithm, it is not necessary to sort the variables and literals, while we can reorder the clauses and the literals in any way to make the searching space smaller. As noted by Adleman [4], the information storage capacity of DNA is huge. In principle, 1 *mmol* of DNA can encode 2 gigabytes of data. The major advantage of DNA computing lies in its high parallelism. Our algorithms take advantage of the high information density and parallel computing capacity of DNA molecules, resembling *in vitro* evolution without generating an initial data pool that contains every possible answer; the number of DNA strands required increase exponentially with the size of the problem, but the observed average and maximum exponent ratio for this algorithm decreases logarithmically with the increasing of the number of variables (n). So our algorithm is more space efficient and can be scaled-up to solve large SAT problems. Unfortunately, the laboratory operations used in this algorithm are still very slow: it takes an average of about 30 min for each operation and 30 h in total to solve a small 3-SAT problem. Although the operations may be further optimized, it is still not yet possible to exceed the performance of electronic computers.

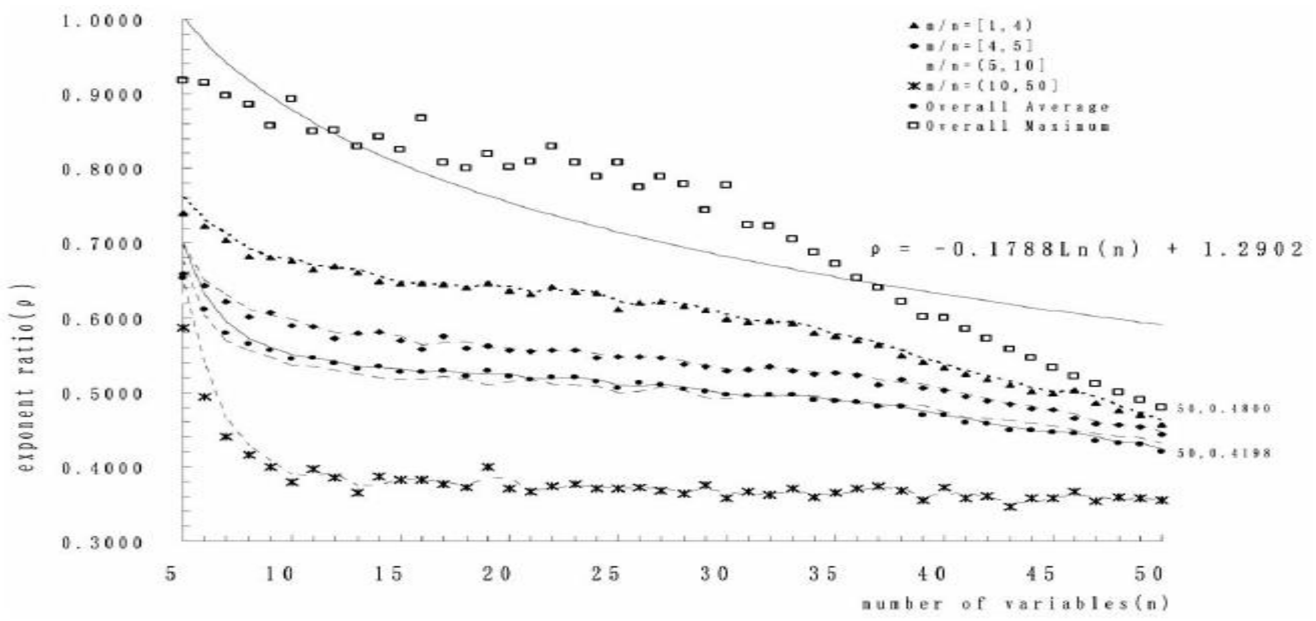

Fig. (4). Average and maximum exponent ratio for different *n* and *m/n* ratio. Data was calculated from fifty thousand random 3CNF–SAT instances with number of variables *n*=(5, 50) and clauses/variable ratio *m/n*=(1, 50).

VI. References

1) Selman B.; Mitchell D.; Levesque H. Generating hard satisfiability problem. *Artif. Intell.*, **1996**, *81*, 17.
2) Wang, X.; Bao, Z.; Hu, J.; Wang, S.; Zhan, A. Solving the SAT problem using a DNA computing algorithm based on ligase chain reaction. *Biosystems*, **2008**, *91*(1), 117.
3) Ouyang, Q.; Kaplan, P.D.; Liu S.; Libchaber, A. DNA solution of the maximal clique problem. *Science*, **1997**, *278*, 446.
4) Adleman, L. Molecular computation of solutions to combinatorial problems. *Science*, **1994**, *266*, 1021.
5) Lipton, R. Using DNA to solve NP-complete problems. *Science*, **1995**, *268*, 542.
6) Yoshida, H.; Suyama, A. DIMACS: Series in Discrete Mathematics and Theoretical Computer Science. Solution to 3-SAT by Breadth-First Search, American Mathematical Society, Providence, **2000**, *RI54*, 9.

7) Li, Y.; Fang, C.; Ouyang, Q. Genetic algorithm in DNA computing: A solution
to the maximal clique problem. *Chinese Sci. Bull.*, **2004**, *49*(9), 967.

8) Liu, W.; Gao, L.; Zhang, Q.; Xu, G.; Zhu, X.; Liu, X.; Xu, J. A random walk DNA algorithm for the 3-SAT problem. *Curr. Nanosci.*, **2005**, *1*, 85.