Solving the 3-SAT Problem Based on DNA Computing

Wenbin Liu,*[†,‡] Lin Gao,[§] Xiangrong Liu,[†] Shudong Wang,[†,‡] and Jin Xu[†]

Department of Control Science and Engineering, Huazhong University of Science and Technology, Wuhan City 430074, China, College of Computers, Xidian University, Xi′an City, 710071, China, and College of Information Science and Engineering, Shandong University of Science and Technology, Taian City 271019, China

Received June 8, 2003

The 3-SAT problem is an NP-complete problem, and many algorithms based on DNA computing have been proposed for solving it since Adleman's pioneering work. This paper presents a new algorithm based on the literal string strategy proposed by Sakamoto et al. Simulation results show that the maximal number of literal strings produced during the computing process is greatly reduced. Moreover, the length of the literal strings is also reduced from *m* to *n* at most.

1. INTRODUCTION

DNA computing is a new vista of computation that bridges between computer science and biochemistry. In DNA based computation, information is carried through DNA molecules, and its processing is accomplished through a series of biochemical reactions. Due to the massive parallelism inherent in the biochemical reactions and the high information density of DNA molecules, DNA computing has become an attractive research field. Since Adleman,[1] and subsequently Lipton,[2] demonstrated the possibility of solving difficult problems such as NP-complete problems, a great deal of research within this field has been focused on the design and implementation of algorithms for those hard problems. Especially, approaches for the Satisfiability (SAT) problem have been investigated frequently, and it has become a benchmark for testing the performance of DNA computers. A group led by Smith in Wisconsin University solved a 4-variable instance SAT problem on a chemically modified Au surface.[3] Later, Wu proposed an improvement to this surface based method by employing an optical film technology so that the surface can be reusable.[4] Yoshida and Suyama solved a 4-variable instance by a breadth first search program implemented with DNA.[5] In addition, Landweber et al. solved an instance of a 9-variable SAT problem related to the "Knights Problem" in chess by RNA.[6] Recently, Adleman's group solved an instance of a 20-variable 3-SAT problem experimentally, and this is the largest instance of the SAT problem solved experimentally by DNA computing.[7]

Notably Sakamoto et al. proposed a unique method to solve the 3-SAT problem. They introduced a literal string strategy implemented through the autonomous hairpin formation of single stranded DNA.[8] The major inefficiency of their method lies in that it increases the solutions space dramatically. To overcome this disadvantage, this paper presents a new algorithm based on the literal string strategy, and the simulation result shows that it can sharply reduce the maximal number of literal strings appearing in the computing process.

2. BACKGROUNDS

2.1. The Satisfiability Problem. Given n Boolean variables $x_1,x_2, \cdots,x_n$, an assignment to those variables is a vector $v = (v_1,v_2, \cdots v_n) \in \{0,1\}^n$. A clause C_i of length k is a disjunction of k literals, $C_i = l_{i1} \vee l_{i2} \vee \cdots \vee l_{ik}$, where a literal is either a variable x_i or its negation $\neg x_i (1 \le i \le n)$. For some constant k, the k-SAT problem is to ask if there is a satisfying assignment that makes a formula $F = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ true. Obviously, there are 2^n possible solutions for such problems. It has been proven that the k-SAT problem is NP-complete for any $k \ge 3$. Throughout this paper, n and m will respectively denote the number of variables and that of the clauses of the input formula F, and in this paper, we are only concerned with the 3-SAT problem in which each clause consists of just 3 literals.

2.2. The Literal String Strategy.[8] For clarity, we give a brief review of the literal strategy proposed by Sakamoto et al., and the formula they solved is presented as

$$F = (x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee x_3 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee x_5) \wedge (x_1 \vee x_4 \vee \neg x_6) \wedge (\neg x_1 \vee x_3 \vee x_4) \wedge (x_1 \vee x_3 \vee \neg x_4) \wedge (\neg x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_3 \vee \neg x_4)$$

A literal string is the conjunction of the literals selected from each clause (one literal per clause), and its length is, m, the number of clauses. A formula is satisfiable if there exists such a literal string that does not contain any conflict literals (this means that, for some literals x_i and $\neg x_i$, they are not allowed to appear in a literal string simultaneously). For example in the formula above, the literal string, $x_2x_3\neg x_4\text{-}\neg x_1x_5\neg x_6\neg x_1x_3\neg x_4\neg x_1$, is a satisfying literal string, and it denotes the only solution of the formula, $F(x_1,x_2,x_3,x_4,x_5,x_6)$

* Corresponding author e-mail: wbliu@mail.hust.edu.cn, wbliu69@sohu.com.
[†] Huazhong University of Science and Technology.
[‡] Shandong University of Science and Technology.
[§] Xidian University.

= (0,1,1,0,1,0). Whereas the literal string, $x_1x_3\neg x_3 \vee x_1x_5 \vee x_6\neg x_1x_3\neg x_4 \vee x_1$, is not because it contains two conflict literals, (x_1,$\neg x_1$) and (x_3,$\neg x_3$). The idea of the literal string strategy is very simple and easy to understand. As the information of the clauses has been transferred into literal strings, each of them can be treated as an independent processor. Under the hierarchy of DNA computing proposed in ref 8, the DNA sequences representing these literal strings will take on different configurations, if a sequence contains no conflict literal, it will present a single linear form; otherwise it will form hairpin structure. Therefore, each of the literal strings represented by DNA molecules functions like a program runs in a computer, and it processes the input data autonomously without any extra intervention. However, its disadvantage is that the solution space of the 3-SAT problem is increased from 2^n to 3^m. Research on the k-SAT problem has shown that the ratio of m/n has a great influence on the possibility of whether the formula is satisfiable or unsatisfiable. More precisely, in the case of $m/n < 4.2$, it is most likely satisfiable; in the case of $m/n > 4.2$, it is most likely unsatisfiable; and in the case of $m/n = 4.2$, it is difficult to decide whether it is satisfiable or not, and for most of the heuristic methods proposed so far, the median computational cost for solving the problem increases dramatically as the problem's size increases in this case.[9] Thus for practical 3-SAT problems, the hardest instances concentrate at the transition point $m/n = 4.2$. Under the literal string strategy, the solution space will become $(3^{4.2}/2)^n \approx 50.5^n$ times that of the standard space 2^n at the transition point. This greatly limits the application of the literal strategy to larger problem instances. Moreover, as the length of the DNA sequences representing the literal strings will increase with the number of clauses rather than that of the variables, this also imposes a limit to the practical application of the literal strategy because DAN sequences longer than 15 000 bases might be fragmented by shear forces of pouring and mixing in test tubes.[10]

2.3. Measures Used To Reduce the Number of Literal Strings. In this paper, three measures are proposed to reduce the maximal number of literal strings during the computation. The main idea is essentially to take advantage of the information provided by the problem itself.

2.3.1. Constructing the Literal String Clause by Clause. In DNA computing, the exponential solution space for most of the NP-complete problems can be synthesized step by step in polynomial time. Thus, it is advisable to adopt this manner for a large instance of problems, instead of that described in Adleman's original paper.[1] Concerning the 3-SAT problem, it allows us to check the partial literal strings that contain conflict literal in each synthesis step and delete them if any. Assuming we have finished the m'th clause and s is a produced partial literal string containing conflict literal, then deleting it means that $3^{m-m'}$ faulty literal strings have been weeded out from the solution pool.

2.3.2. Regrouping the Clauses. It is easy to see that as the earlier partial literal string containing the conflict literals is weeded out the more faulty literal strings will be removed, and we are free to consider them in the following synthesis steps. In DNA based computing, this measure is beneficial to improve the reliability of the biochemical reactions because the faulty solutions have been removed previously.

This idea can be implemented effectively by regrouping the order of the clauses. In fact, the regrouping method has been applied in many heuristic algorithms for the SAT problem though the detailed strategy is different. In this paper, the regrouping process is accomplished through two steps: (1) Assign a mark to each variable according to the frequency of its appearance in all clauses, then calculate the mark for each clause by adding the marks of its variables, and arrange the clauses according to the nonincreasing order. (2) Select the second clause again from the last $m - 1$ clauses so that it includes the most conflict literals with the first clause, then select the third clause from the last $m - 2$ clauses so that it includes the most conflict literals with the first two clauses, and repeat this process until the last clause.

2.3.3. Pruning the Literal Strings. This motivation comes from the observation that for two partial literal strings, $x_1x_4x_1$ and $x_1x_4x_4$, they are treated as different partial literal strings in the original paper of Sakamoto, and both of them will produce at the most 3^{m-3} literal strings, respectively. However, they are essentially equal to the partial literal string x_1x_4, and if they are substituted by x_1x_4, the literal strings produced will be half of that at most. Then the redundance of the literal strings is reduced. This idea can be implemented in the constructing process by checking if the partial literal string includes the current literal; if it does not include the current literal, then add the current literal, otherwise keep it with no change. Additionally, this measure also makes the length of the final literal strings reduced from m to n at most.

2.4. The Algorithm for the 3-SAT Problem. Step 1: regroup the clauses of formula F according to 2.3.2.

Step 2: assuming clause $C_i = c_{i1} \vee c_{i2} \vee c_{i3}$ is being considered and S is the current set of partial literal strings. For each partial literal string $s \in S$, if s includes the conflict literals of $c_{ij}(1 \le j \le 3)$, then no new partial literal string will be produced by it; if s includes literal c_{ij}, then the new partial literal string produced will be itself; otherwise the new partial literal string will be produced by adding c_{ij} at the end of s.

Step 3: repeat step 2 for the next clause until C_m is accomplished.

Step 4: if any literal string remain, say "Yes"; otherwise, say "No".

3. IMPLEMENTATION OF THE ALGORITHM

Considering the first step, it could be accomplished in polynomial time by a greed algorithm through traditional computers. Thus only the other steps are performed by DNA molecules. For each variable $x_i(1 \le i \le n)$, a distinct DNA sequence with fixed length l is used to represent the positive literal x_i, and its complementary sequence represents the negative literal $\neg x_i$. To reduce the false negative in biochemical reactions, those sequences should be designed to keep a low similarity. In addition, we have to prepare $2n$ probe test tubes, upon whose wall a DNA sequence representing a particular literal is attached, so that we can select those literal strings containing a certain literal. The computation process is essentially repeated cycles of constructing all the correct literal strings clause by clause. A mix and split combinatorial synthesis technique can be adopted to synthesize those literal strings. In this technique, single DNA strands representing the combinatorial literal strings are synthesized on small resin beads whose diameter is about 20 μm. The surface of these resin beads is covered with amide groups (NH_2) so that single stranded DNAs can be anchored on it. It is estimated that the surface of a bead can hold about 10^{11} strands.

At the beginning, all the bare resin beads are split into three test tubes. In each tube, one of the three literals in the first clause is synthesized upon the surface of the beads. Then all these resin beads are recombined and mixed for the next cycle. Assuming that the current tube V contains all the DNA strands synthesized from the first $i - 1$ clauses and the current clause is $C_i = c_{i1} \vee c_{i2} \vee c_{i3}$, the following process will implement step 2:

(a) Pour the content of tube V equally into the three probe tubes, say V_{i1}, V_{i2}, and V_{i3} upon whose wall the three literals c_{i1}, c_{i2}, and c_{i3} are respectively attached.

(b) Those partial literal strings, containing the literal conflict with literals c_{i1}, c_{i2}, and c_{i3}, will hybridize with the probes attached on the wall of the three tubes. Then pour the content of V_{i1}, V_{i2}, and V_{i3}, respectively, into the probe tubes V_{i1}', V_{i2}', and V_{i3}' upon whose wall the three literals conflict with c_{i1}, c_{i2}, and c_{i3} are respectively attached. Tubes V_{i1}, V_{i2}, and V_{i3} are regenerated by heating and washing with buffer solution for later using.

(c) Those partial literal strings containing literals c_{i1}, c_{i2}, and c_{i3} will respectively hybridize with the probes attached on the wall of the three tubes and pour the content of V_{i1}', V_{i2}', and V_{i3}', respectively, into tubes V_1, V_2, and V_3.

(d) Add some buffer solution to vessels V_{i1}', V_{i2}', and V_{i3}', heat them to 94 °C, and pour their content to the initial tube V. The content of this tube represents those partial literal strings that contain one of the three literals in the current clause; therefore, the new literal strings are produced by them without change.

(e) The content of the three tubes V_1, V_2, and V_3, respectively, represents these literal strings that contain neither literal c_{i1}, c_{i2}, and c_{i3} nor their negative ones. Simultaneously synthesize literal c_{i1}, c_{i2}, and c_{i3} at the end of DNA strands in tubes V_1, V_2, and V_3, respectively, and pour their content into the tube V.

Finally, if the initial tube V contains conflict-free literal strings, formula F is satisfiable; otherwise it is not.

4. RESULTS AND DISCUSSIONS

To test the performance of our new proposed algorithm, we take a simulation of the construction process of our algorithm for three different 3-SAT problems by computer. The first problem F_1 is the formula listed in section 2.1, the second comes from http://www.w3.org/2002/03owlt/editors-draft/draft/proposed-dl-500-SAT#proposed-dl-500-SAT, and the third is the problem solved by Adleman's group in ref 7. The output ratio is calculated by $\rho = (\log_2^{\mu})/n$, where μ is the number of literal strings. Figures 1−3 show how the number of the literal strings produced changes at each step for the three problems. It is easy to see that the maximal literal strings appeared in the computing process is greatly reduced. For the three problems, the number of their real solution is respectively 1, 2, and 1, while the simulation results of our algorithm shows that the number of final literal strings representing them are respectively 2(24), 48(436), and 3(24) (the number in the parentheses corresponds to that obtained by Sakamoto's original algorithm). Therefore the multiplicity of literal strings representing the final solution of the problems is also reduced.

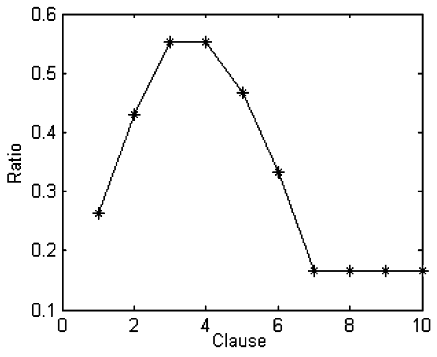

Figure 1. Simulation result for F_1 ($n = 6$, $m = 10$).

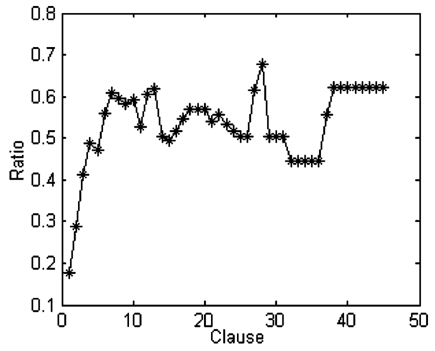

Figure 2. Simulation result for F_2 ($n = 9$, $m = 45$).

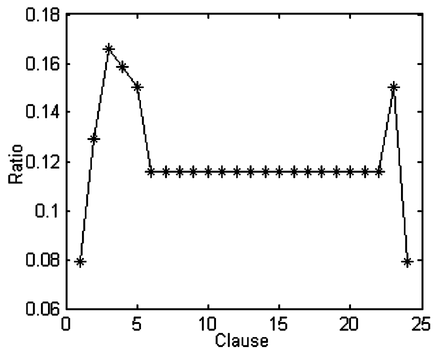

Figure 3. Simulation result for F_2 ($n = 20$, $m = 24$).

Yoshida et al. also introduced a dynamic generation of the candidate solutions for the 3-SAT problem.[5] The candidate solutions are constructed variable by variable, and in each step only one particular variable is considered. For some partial candidate solutions, whether the current variable is assigned to a true value or a false value at its end is determined by how the original input formula F is reduced by it. More precisely, if it reduces the formula F to F' in which the current variable occurs only in a positive form, then the variable is assigned to a true value; and if it occurs in a negative form, then it is assigned to a false value.

Obviously, the main difference between our algorithm and Yoshida's is that the way the information is processed. In each generation step, their algorithm requires that the information included in those clauses containing the current variable have to be checked. Each clause contains three variables, so it has to be checked three times during the generation process. While in our algorithm, the information is processed between the partial literal strings and one of the three literals in the current clause, so it functions as millions of microprocessors work on different input data. This makes the information process in our algorithm easier than that in Yoshida's.

Concerning the solution space, the results of the computer simulation show that Yoshida's algorithm is about $2^{0.5n}$.[5] Although our algorithm also reduces the solution space dramatically compared with Sakamoto's original algorithm in ref 8, it seems to be not enough to beat that of Yoshida's algorithm in this respect. The main reason is because of the redundancy of literal strings. If this drawback of the literal string strategy can be substantially improved, it is promising that this strategy will have a practical application to larger problem instances.

5. CONCLUSIONS

In this paper, we present an improved algorithm based on the literal string strategy, and it has the following advantages: first, the maximal number of literal strings produced during the computing is greatly reduced; second, the length of the literal strings is reduced from *m* to *n*; and third, the main operations used in this algorithm are extraction and synthesis, which are more reliable than the hairpin formation as proposed in ref 8.

However, the multiplicity of literal strings seems to be an intrinsic character of the literal string strategy, and it has become a bottleneck that hinders the practical application of this strategy. Exploiting other heuristic methods to overcome this problem will be the main work in the future.

ACKNOWLEDGMENT

The authors gratefully acknowledge the anonymous reviewers for their constructive suggestions that made this paper a more valuable form. We also thank the authors that appeared in the references.

REFERENCES AND NOTES

(1) Alderman, L. Molecular computations to combinatorial problems. *Science* **1994**, *266*, 1021−1024.
(2) Lipton, R. J. DNA solution of hard computational problems. *Science* **1995**, *268*, 542−545.
(3) Liu, Q. et al. DNA computing on surfaces. *Nature* **2000**, *403*, 175−179.
(4) Wu, H. An improved surface-based method for DNA computation. *Biosystems* **2001**, *59*, 1−5.
(5) Yoshida, H.; Suyama, A. Solution to 3-SAT by breadth first search. DNA Based Computers V, Cambridge, Massachusetts, DIMACS 2000, Vol. 54, pp 9−22.
(6) Cukras, R. et al. Chess games: a model for RNA based computation. *Biosystems* **1999**, *52*, 35−45.
(7) Braich, R.; Chelyapov, N.; Johnson, C.; Rothemund, P.; Adleman, L. Solution of a 20 variable 3-SAT problem on a DNA computer. *Science* **2002**, *296*, 499−502.
(8) Sakamoto, K. et al. Molecular computation by DNA hairpin formation. *Science* **2000**, *288*, 1223−1226.
(9) Selman, B.; Mitchell, D.; Levesque, H. Generating hard satifiability problem. *Artificial Intelligence* **1996**, *81*, 17−29.
(10) Kendrew, J. et al. *The Encyclopedia of Molecular Biology*; Blackwell Science: Oxford, 1994.

CI034113O