A Random Walk DNA Algorithm for the 3-SAT Problem

Wenbin Liu*[,1,2], Lin Gao[3], Qiang Zhang[4], Guandong Xu[1], Xiangou Zhu[1], Xiangrong Liu[2] and Jin Xu[2]

[1]School of Computer Science and Engineering, Wenzhou Normal College, Wenzhou City 325027, China, [2]Department of Control Science and Engineering, Huazhong University of Science and Technology, Wuhan City 430074, China, [3]School of Computer, Xidian University, Xi' an city 710071, China, [4]University Key Lab of Information Science & Engineering, Dalian University, Dalian 116622, China

Abstract: We present a randomized DNA algorithm for the 3-SAT problem based on the probabilistic algorithm proposed by Schöning. The basic idea of our algorithm is that the read of information is performed in linear DNA molecules, while the rewrite information is implemented in plasmid DNAs. Compared with previous works, our algorithm performs the flip of a variable's value more easily and reliably, and the time complexity is also reduced to $O(mn)$, where m is the number of clauses and n is the number of variables. Moreover, Schöning's algorithm has been further improved recently for the case of 3-SAT by Hofmeister. We also demonstrate how to adapt this improvement in our new algorithm and the space complexity of our algorithm is then reduced to $O[(4/3)^{n-3m'} (7/3)^{m'}]$, where m' is the number of the maximal independent clauses. Up to now, this is the most volume-efficient algorithm for the 3-SAT based on DNA computing.

Keywords: DNA Computing, the Satisfiability Problem, Random Walk Strategy, Plasmids DNA.

1. INTRODUCTION

Due to its massive parallelism and high information density, DNA computing has become a new computational paradigm that bridges between computer science and biochemistry. Since Lipton [1] demonstrated the Satisfiability problem could be solved by DNA computing, some other algorithms [2-5] have been proposed for this problem and up to now it has become a benchmark for testing the performance of DNA computing. Apart from the various errors in biochemical reactions, one of the major obstacles for DNA computing is how to cope with the "exponential curse" and reduce the size of the search space. One possible direction is to find algorithms with more efficiency. Ogihara [6] was the first to demonstrate how to implement the algorithm proposed by Monien and Speckenmyer through DNA computing; the space and time complexity of this algorithm is then becomes $O(2^{0.6942n})$and $O(n\cdot\max\{m^2,n\})$ respectively. Chen *et al.* [7] presented a randomized DNA algorithm, based on the classical algori-thm proposed by Paturi for the k – SAT problem, where the space complexity and time complexity is and $O(k^2mn)$ respectively. At present, the most volume efficient algorithm for the 3-SAT is the work presented by Diaz *et al.* [8], where they showed how to implement the random-walk strategy proposed by Schöning with DNA molecules. The space and time complexity of the algorithm then becomes $O((4/3^n)$ and $O(mn^2)$ respectively [9]. The key step of Schöning's method is to select a literal and flip its value. In order to implement this strategy, Diaz introduced a time stamp strategy to denote the state of the current assignments. The two states, "new" and "old", respectively indicate whether an assignment has been updated or not. For each strand representing the specific assignment, its update is realized by adding one copy of its elements at the 3' end except the literal will be flipped. This makes the algorithm too laborious and error-prone because the length of these strands has been doubled in the update processing.

In this paper, we present a new method to implement Schöning's idea. More precisely, the read of information is performed in linear DNA molecules, while its rewrite is finished in plasmids DNA. This makes the flip of a variable's value very easily, and the time complexity is also reduced to $O(mn)$. Additionally, Schöning's algorithm has been improved recently by Hofmeister [10], and we also demonstrate how this improvement could be adapted within the context of our new algorithm.

2. BACKGROUND

2.1. The Satisfiability Problem

Given n Boolean variables x_1, x_2, L, x_n, an assignment to those variables is a vector $v = (v_1, v_2,$ L $v_n)$ $\{0,1\}^n$.A clause C_i of length k is a disjunction of k literals, $C_i = l_{i1} \vee l_{i2} \vee$ L $\vee\ l_{ik}$, 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 asks if there is a satisfying assignment that makes a formula $F = C_1$ C_2 L C_m true. Obviously, there are 2^n possible satisfying assignments for this problem. It has been proven that the k-SAT problem is NP-complete for any $k \ge 3$. Throughout this paper, n will denote the number of variables and m will denote the number of clauses in formula F. And in this paper we are concerned with the 3-SAT in which each clause consists of just 3 literals.

*Address correspondence to this author at the School of Computer Science and Engineering, Wenzhou Normal College, Wenzhou City 325027, China; Tel: 86-0577-88373127; Fax: 86-0577-88373127; E-mail: wbliu@mail.hust.edu.cn and wbliu69@sohu.com

2.2. Schöning's Algorithm [8]

In order to keep this paper self-contained, we briefly review Schöning's random-walk strategy for the 3-SAT problem. It mainly consists of two phases: first, an initial assignment is chosen independently of the clauses; then, pick a literal uniformly at random from one of the clauses that are not satisfied by the current assignment and flip its value. Schöning has proven that a satisfying assignment a^* can always be found with a probability high enough by this strategy at most $3n$ times to $(4/3)^n$ initial assignments. It can be formulated as:

```
Program Search (input: a 3-SAT formula $F$ with $n$ variables)
    For I =1 to $(4/3)^n$
        Select an initial assignment $a$ uniformly at random from $\{0,1\}^n$;
        For J =1 to $3n$
            If $F(a) = 1$ then accept and halt;
            Select a clause $C$ that is not satisfied by $a$ at random;
            Select a literal in $C$ uniformly at random and flip its value in $a$;
        End For J
    End For I
```

Because of the massive parallelism inherent in DNA computing, it has the power to treat $(4/3)^n$ initial assignments simultaneously. Thus we can modify Schöning's algorithm based on DNA computing as follows [9]:

Step 1: Construct $(4/3)^n$ distinct initial assignments for formula F.

Step 2: Detect if there exists any satisfying assignment in the current solution pool. If so, output "YES" and stop. Otherwise, for each assignment a, randomly select a clause that is not satisfied by it and flip one of its three literals' value randomly.

Step 3: Repeat step 2 at most $3n$ times.

2.3. Plasmids [11, 13]

In nature, DNA molecules occur mainly in two forms: linear and circular. The genome of bacteria is usually a circular, double stranded DNA molecule consisting of a few million base pairs. In addition, there exist some extra-chromosomal genetic elements named as plasmids in a variety of species. Plasmids encode information in their sequences to provide for the replication and transcription of their genetic information into RNA within the bacterium in which they occur. Naturally occurring plasmids have been modified to make them more useful as cloning vectors, features including keep a relatively small size, replicating in a relaxed fashion, having selectable markers and containing more unique restriction sites. The position between two adjacent restriction sites is called station, in which interesting sequences can be inserted. Therefore, they can be used to represent information, such as binary variables or vertices of a graph, in DNA computing. Fig. **1** shows a sketch of the kind of plasmids used in this paper.

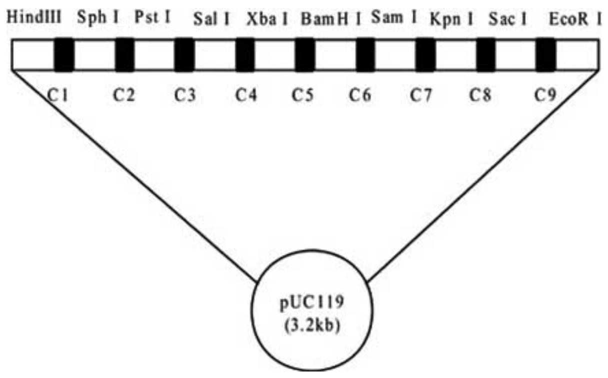

Fig. (1). This figure shows a sketch of the plasmid pUC19. The blank denotes the restriction site recognized by the enzyme beside it, and the black denotes the station. The shadow in station S7 denotes that the recognition sites of the two enzymes are somewhat overlapped, and therefore we can just select one of them to use.

Because the insertion of a sequence into a plasmid is critical to our algorithm, as an example, we give a simple description of this process for station S4 (see Fig. **2**). Assuming that the plasmids are in an appropriate buffer, first, add restriction enzyme *Sal* I to cut the plasmids in the corresponding place; second, add restriction enzyme *Xba* I to

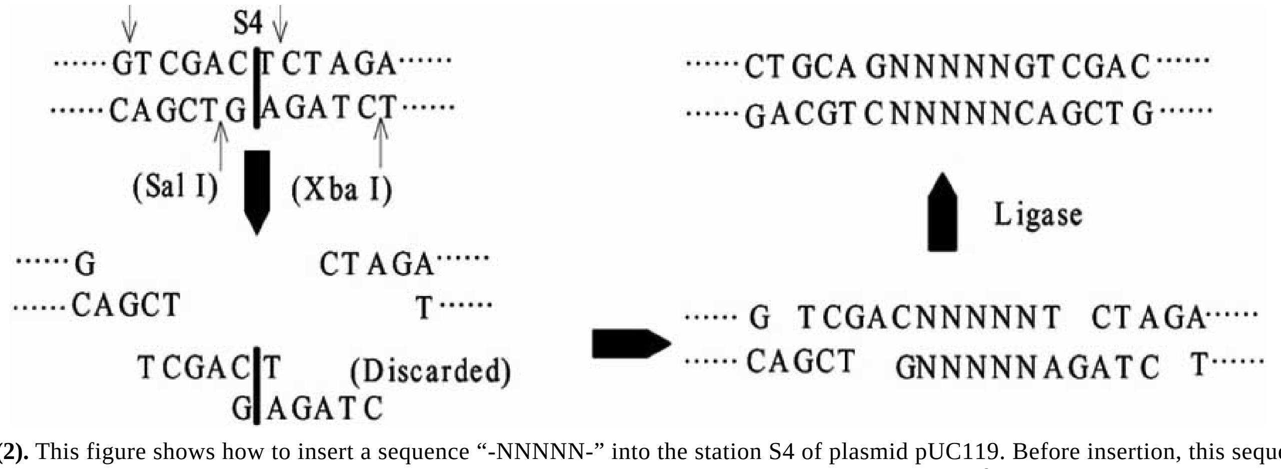

Fig. (2). This figure shows how to insert a sequence "-NNNNN-" into the station S4 of plasmid pUC119. Before insertion, this sequence must be preprocessed so that it carries the sticky ends at both sides, and which will anneal with that of the plasmids.

cut the plasmids in the next recognition site and the segment between the two sites will drop off from the original plasmids; third, clear the dropped sequence by gel electrophoresis, and add the desired sequence which has been preprocessed with the same enzymes as previous steps to the buffer; finally the two pairs of sticky ends will form temporary hydrogen bonds and the ligases will paste the covalent bonds between the inserted sequence and the plasmid to form a circular.

In this paper, we assume that the plasmids are constructed to have similar structure of plasmids pUC19. Therefore they can be denoted as

$$L\, E_0\, S_0\, E_i\, S_1\, E_2\, L_1\, S_1\, E_{i+1}\, L\, E_n\, S_n\, E_{n+1}\, E_{n+2}\, L$$

The subsegments So, S l, L, S n, S n+l denote stations, while the subsegments Eo, E1, L ,En+2 are recognition sites of enzymes REo, REl, L , RE n+2 .

3. IMPLEMENTATION OF THE ALGORITHM

3.1. Encoding Scheme

Let $F = C_1 \vee C_2 \quad L \quad C_m$ be a 3-SAT formula with n variables $x_1,\ x_2\ ,\ L\ , x_n$. Our algorithm requires $2n + 2$well-behaved sequences of DNA. As the encoding problem is an important issue in DNA computing that needs further studies, we shall not give the detailed encodings for them. Here we assume these sequences are designed to have enough difference and stability so that they could be easily distinguished in the computing process.

1. A head sequence h with a fixed length, say 20bp.

2. A tail sequence s with fixed length, say 20bp.

3. n"true" sequences x_i^T with fixed length, say 20bp, representing "x_i = true" .

4. n"false" sequences x_i^F with fixed length, say 12bp, representing "x_i = false".

3.2. Operations

The operations used in this paper are:

1. Denature: break apart a double-stranded DNA into single-stranded complementary components in tube t

2. Double Strand: make each of the single strands in tube t double stranded.

3. Extract: extract from tube t all single strands containing certain particular subsequence.

4. Pour: partition the content of tube t into k test tubes from $u_1$to u_k according the requirements, and tube t then remains empty.

5. Combine: merge the content of tube u_l to u_k into a single tube t.

6. Insert: place certain particular segment into a station of plasmids (as described in section 2.3).

7. Cut: digest plasmids or double stranded DNA molecules with a restriction enzyme.

8. Synthesize: chemically synthesize DNA strands according its encoding.

9. Length: distinguish DNA strands according to their length by gel electrophoresis.

10. Detect: check whether a test tube contains a DNA strand or not.

All these operations are well-studied bio-techniques in genetic engineering [11] and great improvement has been achieved both on their performance both in the speed or accuracy aspect.

3.3. Initial Solution Pool

1. Synthesize sequences $E_0 hE_1$, $E_i x_i^T E_{i+1}$ and $E_{n+1}\ sE_{n+2}$ in different test tubes, and make them double stranded (Concerning the number of the restriction enzymes, we will give a short discussion in the conclusion section). Digest each of them with the first enzyme and then the second sequentially, finally these sequences will become double stranded DNA with particularly sticky ends.

2. Construct the initial test tube t_o with $(4 / 3)^n$ copies of plasmids.

3. Insert sequence $E_o hE_l$ into station S_o in tube t_o by enzymes RE_o and RE_l.

4. Insert sequence $E_{n+1}\ sE_{n+2}$ into station S_{n+1} in tube t_o by enzymes RE_{n+1} and RE_{n+2}.

5. For each i ($l \le I \le n$), pour the content of t_o equally into tubes t_1 and t_2. Then insert sequence $E_i x_i^F E_{i+1}$ into station S_i in tube t_l by enzymes RE_{n+1} and RE_{n+2}. Insert sequence $E_i x_i^T E_{i+1}$ into station S_i in tube t_2 by the same enzymes. Finally combine the content of tube t_1 and tube t_2 into tube t_o.

At this time, tube t_o consists of $(4 / 3)^n$ plasmids, which represent the initial assignments with the following form

$$E_0\, hE_1\, x_1^{T/F} E_2 \cdots E_n\, x_n^{T/F}\ E_{n+1}\, sE_{n+2}$$

It should be mentioned that the length of these segments ranges from $40 + 12n + \sum_{i=0}^{n+2} |E_i|$ to $40 + 20n + \sum_{i=0}^{n+2} |E_i|$ where $|E_i|$ denotes the length of the restriction site E_i.

3.4. Random Walk (this step needs to perform 3*n* times at most)

1. Digest the plasmids in tube t_o with enzyme RE_o completely, and then digest them with enzyme RE_{n+2}. After that the segments representing the initial assignments will drop off from the original plasmids, and we can separate them from the rest of the plasmids by gel electrophoresis. Keep these segments in tube t_l and the rest of the plasmids in tube t_o. Denature the content in tube t_l and extract the single strands containing subsequence $\overline{h}$ and $\overline{s}$, which representing the complementary of subsequences h and s, finally just keep those single strands in tube t_l.

2. Generate a permutation p of integers from 1 to m randomly.

3. For each clause $C_{p(j)} = l_{p(j)1} \quad l_{p(j)2} \quad l_{p(j)3}\ j\ (1 \le j \le m)$, extract from tube t_1 those single strands that fail to satisfy the first literal $l_{p(j)1}$ and put them in tube t_2, then

extract from tube t_2 those fail to satisfy the second literal $l_{p(j)2}$ and put them in tube t_3, finally extract from tube t_3 those fail to satisfy the third literal $l_{p(j)3}$ and put them in tube d_j. At this time tube d_j contains those single strands that fail to satisfy clause $C_{p(j)}$. Combine the content of tube $t_2$and t_3 into tube t_1 for the next clause.

4. As the last clause $C_{p(m)}$. has been investigated, then detect if there is any strand remaining in tube t_1, if so, the formula F is satisfied and stop; otherwise, continue the following steps.

5. For each tube d_j ($l \le j \le m$), perform the length operation to separate these single strands according to their length and regroup them into three test tubes d_{j1}, d_{j2} and d_{j3} randomly. In order to restore these segments into plasmids, first, we make the strands in the $3m$ tubes double stranded, then add suitable DNA adapters so that they can anneal with the sticky end of strands in tube t_o to form the circulated plasmids. (Concerning the DNA adapters, it is somewhat complicated and readers are recommended to refer [13]).

6. This step will flip a literal's value. For each tube d_{ji} ($l \le j \le m$, $l \le i \le 3$):

(a) Add the double stranded DNA in tube d_{ji} to tube t_o, then some of the plasmids will restore to circular form while others still remain to be linear.

(b) Separate those circulated plasmids from the linear by gel electrophoresis, and keep the former in tube d_{ji} and the latter in tube t_o.

(c) In tube d_{ji}, insert the segments representing the negative of literal $l_{p(j)i}$ into station $S_{p(j)i}$

Finally, combine the content of those $3m$ tubes (d_{11},L,d_{m3}) into tube t_o, which represents the refreshed solution pool.

4. ANALYSIS OF THE ALGORITHM

To implement Schöning's algorithm with DNA computing, it is important to keep randomly select a literal that intend to will be flipped. In our algorithm, this is accomplished through two strategies: First, the clauses are selected randomly. A specific assignment a may fail to satisfy several clauses at the same time, thus the selection of a clause has a great influence on which literal is selected to flip. Second, the positive and negative literals of variable x_i are encoded with different length, 20bp and 12bp. Thus the diversity in length allows us to allocate those fail to satisfy the same clause into three tubes, and just select one literal to be flipped in each tube.

Now we give a simple analysis of the running time by the number of sequential operations. The construction of initial solution pool, each variable x_i needs one pour operation, two cut operations, two insert operations and one combine operation. In addition, the head sequence h and the tail sequence s need two cut operations and two insert operations. The total operations of this process are $6n$+4, and its time complexity is $O(n)$.

Concerning the random walk process, first, to make these initial assignments become single strands, we have to perform a cut operation, a denature operation and two extract operations. Next, to decide which literal will be flipped, we have to perform three extract operations and one length operation for each clause C_j ($l \le j \le m$). As all clauses have been investigated, a detect operation is needed to check if there exists a satisfying assignment. And then, to flip the value of the selected literals, we have to perform one double strand operation, one length operation, two insert operations and one cut operation for each of the $3m$ test tubes. In the end, one combine operation is performed to collect these refreshed assignments into tube t_o. Therefore, each cycle of the random walk takes at most $19m$+ 6 operations. Since the random walk is performed at most $3n$ times in our algorithm, then the total operations is at most $3n(19m + 6)$. Therefore the time complexity of our algorithm becomes $O(mn)$.

5. IMPROVEMENT OF THE ABOVE ALGORITHM

In Schöning's original algorithm, the initial assignment is selected randomly without considering the information hidden in the clauses. Each variable is set to 0 or 1 with equal probability. Hofmeister [10] proposed to make some improvement in the generation of the initial assignment. His idea comes from the observation that if there is a clause $C = x_i \quad x_j \quad x_k$, then the three variables should not be set to 0 simultaneously in a satisfying assignment a^*. In order to use this information, he introduced a concept of the independent clause set. Two clauses C_i and C_j are independent if they share no common variables. A maximal independent clause set F' is a subset of clauses in F such that all clauses in F' are independent and every other clause in F shares some variables with some clause of F'. Assume that m' is the number of clauses in F', and those independent clauses are denoted as C_1,L ,$C_{m'}$. By renaming variables and exchanging the role of x_i and $\overline{x_i}$ if necessary, we can assume that variables x_1, x_2,L,$x_{3m'}$ are those contained in the independent clauses, and $x_{3m'+1}$,L, x_n are the remaining variables. In addition, all clauses in F' have the positive form

$$C_1 = x_1 \vee x_2 \vee x_3$$

$$C_2 = x_4 \vee x_5 \vee x_6$$

$$\cdots$$

$$C_{m'} = x_{3m'-2} \vee x_{3m'-1} \vee x_{3m'}$$

Based on this observation, Hofmeister proposed the following strategy to assign values for the $3m'$ variables in the independent clause set F'.

```
Program Assign ($p_1$,$p_2$ ,$p_3$)
    % $p_i$ are the probabilities with $3p_1+3p_2+3p_3 = 1$ for $C_i = x_{3m'-2} \vee x_{3m'-1} \vee x_{3m'}$ ($l \le i \le m'$).
For I =1 to $m'$
    Set values for the three variables in $C_i$ such that for each $c = (x_{3i-2}, x_{3i-1}, x_{3i})$ the following holds:
    Pr {$c$ = (0,0,1) }= Pr {$c$ = (0,1,0) }= Pr {$c$ = (1,1,0) }= $p_1$;
    Pr {$c$ = (0,1,1) }= Pr {$c$ = (1,0,1) }= Pr {$c$ = (1,1,0) }= $p_2$;
    Pr {$c$ = (1,1,1) }= $p_3$;
End For I
```

Hofmeister has proved that the time complexity of Schöning's algorithm can be reduced from $(4/3)^n$ to $(4/3)^{n-3m'}(7/3)^{m'}$ ($1 \le m' \le n/3$) as p_1, p_2 and p_3 are assigned to 4/21, 2 / 21 and 3/ 21. Obviously, this improvement depends on the value of *m*', which is at most *n* / 3 for the 3-SAT problem. In order to incorporate this idea to our new algorithm, we only need to make some modifications to assign the first 3*m*'variables:

(a) Pour the contents of t_0 into tubes t_{10} and t_{11} according to the fractions indicated in Fig. **3**. Insert sequence $E_{3i-2}x^{F}_{3i-2}E_{3i-1}$ ($1 \le i \le m'$) in station S_{3i-2} in tube t_{10} and $E_{3i-2}x^{T}_{3i-2}E_{3i-1}$ in that of tube t_{11}.

(b) Pour the contents of tube t_{10} into tubes t_{20} and t_{21}, and that of tube t_{11} into tubes t_{22} and t_{23} according to the fractions indicated in Fig. **3**. Insert sequences $E_{3i-1}x^{F}_{3i-1}E_{3i}$ in station S_{3i-1} in tubes t_{20} and t_{22}, while insert sequences $E_{3i-1}x^{T}_{3i-1}E_{3i}$ in that of tubes t_{21} and t_{23}.

(c) Pour the content of tube t_{20} into tube t_{30}, the content of tube t_{21} into tubes t_{31} and t_{32}, the content of tube t_{22} into tubes t_{33} and t_{34}, and that of tube t_{23} into tubes t_{35} and t_{36} according to the fractions indicated in Fig. **2**. Insert sequences $E_{3i}x^{F}_{3i}E_{3i+1}$ in station S_{3i} in tubes t_{31}, t_{33} and t_{35}, while insert sequences $E_{3i}x^{T}_{3i}E_{3i+1}$ in that of tubes t_{30}, t_{32}, t_{34} and t_{36}.

(d) Finally combine the content of tubes from t_{30} to t_{36} into tube t_0.

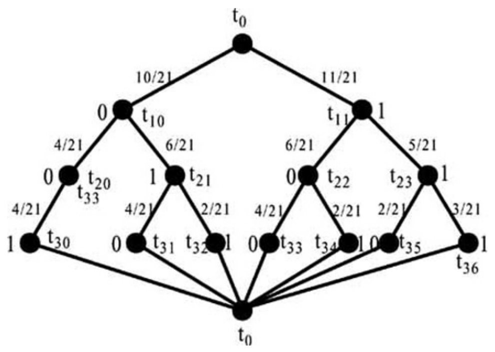

Fig. (3). This figure shows the process of synthesizing the three variables appeared in an independent clause. Each of the three middle layers represents just one variable is assigned. Each point denotes a test tube and the fractions beside lines indicate the fraction of the previous tube to be poured to the next two tubes. Symbols "0" and "1" beside the point indicate the value that the current variable is assigned.

Obviously, we need to perform 7 pour operations, 13 insert operations, 13 cut operations and one combine operation to assign values for the variables in one independent clause. It sums up to 34*m*' for the first 3*m*'variables while $6(n - 3m') + 4$ operations are needed for the rest. Then adopting Hofmeister's strategy in the construction process makes the number of operations increase from 6n + 4 to $6n + 4 + 16m'$, which is at most $12n + 4$ given. $1 \le m' \le n/3$ Thus the time complexity of this process still keeps $O(n)$.

CONCLUSION

In this paper, we present a new method to implement Schöning's random-walk strategy for the 3-SAT problem based on DNA computing. This strategy is very simple and just includes two steps: select a literal and flip its value. Our idea is that the selection of a literal is accomplished in linear DNA molecules, and the flip of its value is performed in plasmid DNA. As the circular form of plasmids makes it more suitable to remove or insert a segment of DNA (corresponding to the flip operation) than linear DNA molecules, our algorithm takes less time than that proposed in [9]. Furthermore, the performance of flip process is more reliable. In addition, we also show how to incorporate Hofmeister's idea into our algorithm.

However, the main drawback of our algorithm is that there are so many enzymes involved in the computation process. First, the instance of the problem is limited by the number of enzymes that can be used without confliction; the other is how to maintain a high efficiency of the enzymes as they act in the same buffer. In order to overcome this disadvantage, a specifically designed protein nucleic acid (PNA) may be a promising technique, which can suppress restriction of particular restriction sites and it may be possible to use the same restriction enzyme for multiple stations in the future [12]. In addition to that, the uncertainty of the real biochemical reaction may also have a great influence on the performance of our algorithm. Therefore, great improvements in the biochemical techniques are urgently desired.

ACKNOWLEDGEMENTS

This paper is supported by the National Scientific Foundation of China under grants 60403002, 60403001 and 30470414. The authors are gratefully acknowledged to the anonymous reviewers for their constructive suggestions that make this paper a more valuable form. And thanks also to the authors that appeared in the references.

REFERENCES

[1] Lipton, R. J. *Science*, **1995**, *268*, 542-545
[2] Liu, Q.; Wang, L.; Frutos, A.G.; Condon, A.E.; Corn, R.M.; Smith, L.M. *Nature*, **2000**, *403*,175-179.
[3] Wu, H. *Biosystems*, **2001**, *59*,1-5.
[4] Cukras, A.R.; Faulhammer, D.; Lipton, R.J.; Landweber, L.F. *Biosystems*, **1999,** *52*,35-45.
[5] Braich, R.; Chelyapov, N.; Johnson, C.; Rothemund, P.; Adleman, L. *Science*, **2002**, *296*, 499-502.
[6] Ogihara, M. *Technical Report TR 629*, University of Rochester, Department of Computer Science, Rochester, NY, July **1996**.
[7] Chen K.; Ramachandran, V. *6th International Workshop on DNA-Based Computers*, Lecture Notes in Computer Science, **2000**, *2054*,199-208.
[8] Schöning, U. *In Proceedings of 40th Symposium on Foundations of Computer Science*, IEEE Computer Society Press, **1999**, 410–414.
[9] Díaz, S.; Esteban, J.; Ogihara, M. *6th International Workshop on DNA-Based Computers, Lecture Notes in Computer Science,* **2000,** *2054*, 209-219.
[10] Hofmeister, T.; Schöning, U.; Schuler R.; Watanabe, O. *Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, **2002**, 192-202.
[11] Kendrew, J. *The Encyclopedia of Molecular Biology*, Blackwell Science: Oxford, **1994.**

[12] Nielsen, P.; Egholm, M.; Berg, R.; Buchardt, O. *Nucleic Acids Research*, **1992,** *21*, 197–200.

[13] Meyers, R. *Molecular Biology and Biotechnology-A Comprehensive Desk Reference*, VCH Publishers Inc: New York, **1995**.

Received: 20 August, 2004 Accepted: 07 October, 2004