

一种改进的最大团问题 DNA 计算机算法

李肯立^{1),2)} 周 旭¹⁾ 邹舒婷¹⁾

¹⁾(湖南大学计算机与通信学院 长沙 410082)

²⁾(华中科技大学分子生物计算机研究所 武汉 430074)

摘 要 随着 DNA 计算的不断发展,如何克服穷举算法带来的指数爆炸问题已成为 DNA 计算领域的重要研究目标之一.将图灵机中的剪枝算法设计技术应用于最大团问题的 DNA 计算中,提出一种最大团问题的新 DNA 计算机算法.算法由顶点度数搜索器、团生成器、稀疏图与稠密图并行搜索器以及最大团搜索器组成.与已有文献同类算法的对比分析表明:文中算法在保持多项式操作时间的条件下,将求解 n 个顶点的最大团问题所需 DNA 分子链数从现有文献的 $O(2^n)$ 减少至 $O(\sqrt{3}^n)$,同时文中算法还具有高效的空間利用率及容错能力的优点.

关键词 DNA 超级计算;最大团问题;剪枝技术;NP 完全问题

中图法分类号 TP301

Improved Volume Molecular Solutions for the Maximum Clique Problem on DNA-Based Supercomputing

LI Ken-Li^{1),2)} ZHOU Xu¹⁾ ZOU Shu-Ting¹⁾

¹⁾(School of Computer and Communications, Hunan University, Changsha 410082)

²⁾(Institute of Biomolecular Computer, Huazhong University of Science and Technology, Wuhan 430074)

Abstract How to decrease the volume in DNA computers is of a great significance in the research of DNA computing. For the objective to decrease the DNA volume of the maximum clique problem which is a famous NP-complete problem, the pruning strategy is introduced into the DNA supercomputing and a new DNA algorithm is proposed. The new algorithm consists of a Degree Searcher, a Clique Generator, a Dense Parallel Searcher, a Sparse Parallel Searcher and a Maximum Clique Searcher. In a computer simulation, the DNA strands of maximum number required is $O(\sqrt{3}^n)$ on the condition of not varying the time complexity where n is the number of the vertex in a graph, while $O(2^n)$ DNA strands are needed in present molecular algorithms for the maximum clique problems. Furthermore, this algorithm is highly space-efficient and error-tolerant compared to conventional brute-force searching.

Keywords DNA-based supercomputing; maximum clique problem; pruning strategy; NP-complete problem

1 Introduction

DNA computing is a computational model that uses bio-molecules as information storage materials

and biological laboratory experiments as information processing operators^[1]. In 1994 Adleman^[2] succeeded to solve an instance of the Hamiltonian path problem in a test tube, just by handling DNA

strands. The power of parallel, high density computation by molecules in solution allows DNA computers to solve hard computational problems such as NP-complete problems in polynomially increasing time, while a conventional Turing machine requires exponentially increasing time. However, most of the current DNA computing strategies are based on enumerating all candidate solutions. These algorithms require that the size of the initial data pool increases exponentially with the number of variables in the calculation, so that the capacity of the DNA computer is limited. And what is more, Fu^[3] presented the enumeration algorithms made the length may also too long to make the algorithm to be length-efficient.

In order to break the barrier of this kind of simply enumerate method, Bach et al^[4] proposed a $n1.89^n$ volume, $O(n^2 + m^2)$ time molecular algorithm for the 3-coloring problem and a 1.51^n volume, $O(n^2 m^2)$ time molecular algorithm for the independent set problem, where n and m are, subsequently, the number of vertices and the number of edges in the problems resolved. Fu^[5] presented a polynomial-time algorithm with a 1.497^n volume for the 3-SAT problem, a polynomial-time algorithm with a 1.345^n volume for the 3-coloring problem and a polynomial-time algorithm with a 1.229^n volume for the independent set. Though the size of those volumes^[4-5] is lower, constructing those volumes is more difficult and the time complexity is higher.

Maximum Clique Problem^[6], which is widely used in the information retrieval, experimental design and computer vision, is a famous combinatorial optimization problem. Now the algorithm for the maximum clique problem can not meet the needs of the application.

In this paper, we describe an algorithm to solve the maximum clique problem. Since Qu's^[7] paradigm proposed in 2004 demonstrated the feasibility of applying DNA computer to tackle such an NP-complete problem. Instead of surveying all possible assignment sequences generated at the very beginning, we use the operations of Adleman-Lipton model and the solution space of sticker which is proposed by Chang et al., then apply the pruning strategy, a new DNA algorithm for maximum clique problem is proposed. The proposed algorithm can solve the maximum clique problem by using the $O(\sqrt{3}^n)$ shorter DNA strands without substantially increasing their time complexity.

2 Reference Model

Chang et al.^[8] presented the model that took biological operations in the Adleman-Lipton model and the solution space of stickers in the sticker-based model^[9-10]. This model has several advantages from the Adleman-Lipton model and the sticker-based model:

(1) The new model has finished all the basic mathematical functions, and the number of tubes, the longest length of DNA library strands, the number of DNA library strands and the number of biological operations are polynomial.

(2) The basic biological operations in the Adleman-Lipton model had been performed in a fully automated manner in their lab. The full automation manner is essential not only for the speedup of computation but also for error-free computation.

(3) Chang and Guo^[11-12] also employed the sticker-based model and the Adleman-Lipton model for dealing with Cook's theorem, the set-packing and clique problems, the subset-product problem and many other NP complete problems for decreasing the error rate of hybridization.

If given a tube, one can perform the following operations in Adleman-Lipton model.

3 Improved DNA Algorithm for Solving the Maximum Clique Problem

The maximum clique problem is an NP-complete problem in computer science^[13]. Let $G=(V, E)$ be a graph with n nodes. A clique in G is defined as a subset $V' \subset V$ such that each two vertices in V' are connected by an arc in E . The Maximum Clique Problem then involves finding the biggest subset V' of totally connected nodes in the graph.

The main principle of the improved DNA algorithm for the maximum clique problem is the following:

Step1. Preprocessing the maximum clique problem and searching the vertexes whose degrees are " $n-1$ " or " 0 ".

Step2. Producing the resolvent solution spaces for the maximum clique problem.

Step3. Using the pruning strategy for eliminating unfeasible solutions from the resolvent solution spaces.

Step4. Basing on step3, producing the solution spaces of the maximum clique problem.

Step5. Searching for satisfiable solution space.

Step6. Finding the solutions of the maximum clique problem.

The improved DNA algorithm corresponding to the basis algorithm above is as follows:

Let $G = (V, E)$ be a finite undirected graph with vertex set $V = \{v_1, v_2, \dots, v_n\}$ and edge set $E = \{e_1, e_2, \dots, e_n\}$. The i th edge is denoted by $e_i = \{v_{i1}, v_{i2}\}$.

3.1 Construction of a Parallel Degree Searcher

Before producing the resolvent solution spaces, the parallel degree searcher is designed to search the vertexes whose degrees are “0” or “ $n-1$ ”. The vertexes whose degrees are “0” are not in the solution of the maximum problem, and the vertexes whose degrees are “ $n-1$ ” are always in the solution of the maximum problem.

Procedure. Degree_Searcher(E, V, n).

1. For $i=n$ down to 1
2. If (Degree(v_i)= $i-1$) then
 - v_i is the vertex with the maximum degree in the Graphics
 - 2a. $E' = E - E_i$ and $V' = V - v_i$.
 - 2b. Degree_Searcher($E', V', i-1$).
- Else
3. If (Degree(v_i)=0) then
- 3a. $V' = V - v_i$.
- 3b. Degree_Searcher($E', V', i-1$).
- EndIf
- EndIf
- EndFor
- EndProcedure

The algorithm, Degree_Searcher(E, V, n) can be used for preprocessing the maximum clique problem. Degree_Searcher(E, V, n) is a recursive procedure. Searching the vertex v_i whose degree is “0” or “ $n-1$ ” and eliminating v_i and the edges corresponding to it e_i from the graph G . At last we can get a new graph G_1 . So finding the maximum clique of the graph G is translated to finding the maximum clique of the graph G_1 .

3.2 DNA Algorithm for Resolvent Solution Space of the Maximum Clique Problem

This algorithm for constructing the resolvent solution space of the maximum clique problem is similar to the Michael’s algorithm^[11-12].

First we find the vertex v_j which has the minimization degree of the vertexes in the graph G_1 . Suppose that there is $k-1$ vertexes adjoining to v_j . Assume that $x_1 \dots x_k$ is an k -bit binary number, which is applied to represent k vertexes. For convenience, we assume that x_i^1 denotes the vertex v_i that is in the maximum clique problem and x_i^0 denotes the vertex v_i that is not in the maximum

clique problem. For example, $x_1^0 \dots x_k^0$ represents that the vertexes $v_i (0 \leq i \leq k)$ are not in the maximum clique problem. In this process, we will consider k vertexes for producing the resolvent solution, where the k vertexes are chosen differently due to the graph G_1 .

Procedure. Init_Resolvent_Solution(T_0, k)

1. For $m=1$ to k
 - 1a. Amplify (T_0, T_1, T_2).
 - 1b. Append (T_1, x_m^1).
 - 1c. Append (T_2, x_m^0).
 - 1d. $T_0 = \cup(T_1, T_2)$.

EndFor

EndProcedure

Lemma 1. The resolvent solution space of 2^k possible arrays of input can be constructed with sticker in a sticker-based model from the algorithm, Init_Resolvent_Solution(T_0, k).

Proof. Assume that T_0, T_1 and T_2 are distinct test tubes but only T_1 and T_2 are empty. The algorithm, Init_Resolvent_Solution(T_0, k) is implemented via the amplifying, append and merge operations. Each time step 1a is used to amplify tube T_0 and to generate two new tubes, T_1 and T_2 , which are copies of T_0 . Tube T_0 becomes empty. Then, step 1b is applied to append a DNA sequence (sticker), representing the value “1” for x_m , onto the end of every strand in tube T_1 . Step 1c is also employed to append a DNA sequence (sticker), representing the value “0” for x_m , onto the end of every strand in tube T_2 . Next, step 1d is used to pour tube T_1 and T_2 into tube T_0 . This indicates that DNA strands in tube T_0 include DNA sequences of $x_m=1$ and $x_m=0$. After repeating the execution of step 1a through step 1d, it finally produces tube T_0 that consists of 2^k DNA sequences representing 2^k possible arrays of input. Therefore, it is inferred that sticker-based solution space for 2^k possible arrays of input can be constructed with sticker. \square

From Init_Resolvent_Solution(T_0, k), it takes k amplify operations, $2 \times k$ append operations, k merge operations and three test tubes to construct sticker-based solution space. A k -bit binary number corresponds to an array of input. A value sequence for every bit contains 15 bases. Therefore, the length of a DNA strand, encoding a subset, is $15 \times k$ bases consisting of the concatenation of one value sequence for each bit.

3.3 Construction of a Clique Generator

In order to reduce the DNA strands of the maximum clique problem, a parallel clique generator is designed.

Procedure. Clique_Generator(T_0, n, E', V)

```

1. For  $i=k+1$  to  $n$ 
  Stop1;
2. For  $j=1$  to  $i-1$ 
  2a. If ( $e_{ij}=0$ )
    2b.  $T_1 = + (T_0, x_j^1)$ .
    2c. If (Detect ( $T_1$ ) = "yes") then
      2d. Append ( $T_1, x_i^0$ ).
      2e. break Stop 1;
  EndIf
EndFor
3. Amplify ( $T_0, T_2, T_3$ ).
4. Append ( $T_2, x_i^1$ ).
5. Append ( $T_3, x_i^0$ ).
6.  $T_0 = \cup (T_1, T_2, T_3)$ .
EndFor
EndProcedure

```

Lemma 2. The algorithm, Clique_Generator (T_0, n, E', V) can be applied to construct the solution space of the maximum clique problem.

Proof. By the algorithm, Init_Resolvent_Solution(T_0, k), we can gain the resolvent solution space. After eliminating the illegal solutions, we can get the solution space with the algorithm Clique_Generator(T_0, n, E', V). Step 2 is a inner loop and is mainly employed to produce the solution space. Step 2a employs the detect operations to detect whether v_i and v_j is adjoin. If v_i is adjoin to v_j , step 2a uses the extract operation to form a test tubes: T_1 . Tube T_1 includes all of the strands that have $x_j = 1$. On the execution of step 2c, the detect operation is applied to check if tube T_1 contains DNA strands. If it returns a 'yes', then step 2d through step 2e are run. On the execution of step 2d, it uses the append operation to append 15-based DNA sequences for representing x_i^0 onto the tail of every strand in T_1 . On the execution of step 2e, this inner loop will be broke and a new inner loop starts. Step 3 is used to amplify tube T_0 and to generate two new tubes, T_2 and T_3 , which are copies of T_0 . Tube T_0 becomes empty. On the execution of step 4, it uses the append operation to append 15-based DNA sequences for representing the value "0" of x_i onto the tail of every strand in T_2 . On the step 5, it uses the append operation to append 15-based DNA sequences for representing x_i^0 onto the tail of every strand in T_3 . \square

From Clique_Generator(T_0, n, E', V), it takes $(n-k-1)$ extract operations, $(n-k-1)$ amplify operations, $2 \times (n-k-1)$ append operations, $(n-k-1)$ merge operations and 4 test tubes. The length of a DNA strand is $15 \times n$.

After the algorithms, Init_Resolvent_Solution (T_0, k) and Clique_Generator (T_0, n, E', V), the

solution space for the graph G will be produced.

3.4 Construction of a Solution Searcher

The Michael's algorithm^[11] and Quyang's algorithm^[13] for maximum clique problem, they all applied brute force algorithm in the search process. Every vertex must check whether it has an edge with the left $n-1$ vertexes in the complement of the graph.

In this paper, we design a sparse graph searcher and a dense graph searcher.

3.4.1 DNA based algorithm for a sparse solution searcher

In a sparse graph, assume that e is an edge in \bar{G} and $e = (v_c, v_d)$, where \bar{G} is the complement of the graph G , the library strands with $x_c = 1$ and $x_d = 1$ are illegal strands. p is used to represent the number of edges considered in the sparse parallel searcher. The sparse graph's parallel searcher is constructed as follows.

Procedure. Sparse_Parallel_Searcher(T_0, n, E', V)

```

1. For  $i=1$  to  $|E'|$ 
  Assume that  $e_m$  is an edge in  $\bar{G}$  and  $e_m = (v_c, v_d)$ 
  1a.  $T_1 = + (T_0, x_c^1)$  and  $T_2 = - (T_0, x_c^1)$ .
  1b. If (Detect( $T_1$ ) = "yes") then
    1b0.  $T_3 = + (T_1, x_d^1)$  and  $T_4 = - (T_1, x_d^1)$ .
    1b1. If (Detect ( $T_3$ ) = "yes") then
      1b2. Discard ( $T_3$ ).
    Else
    1b3.  $T_0 = \cup (T_2, T_4)$ .
  EndIf
  Else
     $T_0 = \cup (T_0, T_2)$ .
  EndIf
EndFor
EndProcedure

```

Lemma 3. The algorithm, Sparse_Parallel_Searcher(T_0, n, E', V) can be applied to search the solution of the clique problem from the sparse graph's solution space.

Proof. The algorithm, Sparse_Parallel_Searcher(T_0, n, E', V), is implemented by means of the extract, detect and merge operations. Step 1 is the outer loop. On the first execution of step 1a, it uses the extract operation to form two test tubes: T_1 and T_2 . Tube T_1 includes all of the strands that have $x_c = 1$. Tube T_2 consists all of the strands that have $x_c = 0$. On the execution of step 1b, the tube T_1 is first detected with "detection" operation. If it returns "yes", then the tube contain the vertex v_c and step 1b₀ ~ 1b₃ are run. When the first execution of step 1b₀ uses the extract operation to form two test tubes: T_3, T_4 . The first tube contains all of the strands that have $x_d = 1$, that is to say, the vertex v_d appears in tube

T_3 . The second tube T_4 includes all of the strands that have $x_d=0$, that is to say, the vertex does not occur in tube T_4 . On the execution of step $1b_1$, it uses the detect operation to test is there is any DNA sequence in tube T_3 . If it returns a “yes”, this indicates that the vertex v_c, v_d appear in the tube T_3 , then on the execution of step $1b_2$, it uses discard operation to discard the tube T_3 . If it returns “no”, step $1b_3$ will be run. It is indicated that the tube T_2 , and T_4 contains the strands, which satisfy the definition of maximum clique problem. Lastly, the execution of step $1b_4$ applies the merge operation to pour tubes T_2 into T_0 . Repeat execution of step $1a$ and step $1b_4$ until every vertex in the graph G are considered. All the legal solution strands will be contained in tube T_0 . \square

From Sparse_Parallel_Searcher(T_0, n, E', V), it takes $2 \times p$ exact operations, $2 \times p$ merge operations, 2 detect operations and five test tubes.

3.4.2 DNA based algorithm for a dense solution searcher

In the dense graph, assume that a strand with $x_c=1, x_d=1$, if in the graph \bar{G} there is an edge between the vertex v_c and v_d , the strand is illegal. n_0 is assumed to represent the number of the vertices considered in the dense parallel searcher. The dense graph's parallel searcher is constructed as follows.

Procedure. Dense_Parallel_Searcher(T_0, n_0, E', V)

1. For $i=1$ to n_0-1
 - 1a. $T_1 = +(T_0, x_i^1)$ and $T_2 = -(T_0, x_i^1)$.
 - 1b. If (Detect(T_1) = “yes”) then
 - 1b₀. For $j=i+1$ to n_0
 - 1b₁. $T_3 = +(T_1, x_j^1)$ and $T_4 = -(T_1, x_j^1)$.
Assume that e is an edge in \bar{G} and $e = (v_i, v_j)$
 - 1b₂. Discard(T_3).
 - EndFor
 - 1c. $T_0 = \cup(T_2, T_4)$.
Else
 - 1d. $T_0 = \cup(T_0, T_2)$.
EndIf
- EndFor
- EndProcedure

Lemma 4. The algorithm, Dense_Parallel_Searcher(T_0, n_0, E', V) can be applied to search the solution of the clique problem from the dense graph's solution space.

Proof. The algorithm, Dense_Parallel_Searcher(T_0, n_0, E', V), is implemented by means of the extract, detect and merge operations. Step 1 is the outer loop. On the first execution of step 1a, it uses the extract operation to form two test tubes: T_1 and T_2 . Tube T_1 includes all of the

strands that have $x_i=1$. Tube T_2 consists all of the strands that have $x_i=0$. On the execution of step 1b, the tube T_1 is first detected with “detection” operation. If it returns “yes”, then step $1b_0 \sim 1b_4$ will be run. Step $1b_0$ is an inner loop. When the first execution of step $1b_1$, it uses the extract operation to form two test tubes: T_3, T_4 . The first tube T_3 contains all of the strands that have $x_j=1$, that is to say, the vertex v_j appears in tube T_3 . The second tube T_4 includes all of the strands that have $x_j=0$, that is to say, the vertex v_j does not occur in tube T_4 . Assume that there is an edge between the vertexes v_i and v_j , tube T_3 contains all the illegal solution strands. On the execution of step $1b_2$, it uses discard operation to discard the tube T_3 . Step 1c applies merge operation to pour tube T_2 and T_4 into tube T_0 . It is indicated that the tube T_2 and T_4 contains the strands, which satisfy the definition of maximum clique problem. The execution of step 1c applies the merge operation to pour tubes T_2 and T_4 into T_0 . If step 1b returns “no”, step 1d will be run. On the execution of step 1d, it applies the merge operation to pour tube T_2 into tube T_0 . Repeat execution of step 1a and step 1d until every vertex in the graph G are considered. All the legal solution strands will be contained in tube T_0 . \square

From Dense_Parallel_Searcher(T_0, n, E', V), it takes $(n_0-1) \times (n_0-1)/2$ exact operations and (n_0-1) merge operations, $(n_0-2) \times (n_0-1)/2$ discard operations, (n_0-1) detect operations, and five test tubes.

3.5 The Construction of a Maximum Clique Searcher

For searching the solution of the maximum clique, we produce a maximum clique searcher.

Procedure. Maximum_Clique_Searcher(T_0, n)

1. For $i=0$ to $n-1$
 2. For $j=i$ down to 0
 - 2a. $T_{j+1}^{\text{on}} = +(T_j, x_{i+1}^1)$ and $T_{j+1}^{\text{off}} = -(T_j, x_{i+1}^1)$.
 - 2b. $T_{j+1} = \cup(T_{j+1}, T_{j+1}^{\text{on}})$.
 - EndFor
- EndFor
3. For $k=n$ down to 1
 - 3a. If (Detect(T_k) = “yes”) then
 - 3b. Read(T_k) and terminate the algorithm.
 - EndIf
- EndFor
- EndProcedure

Lemma 5. The algorithm, Maximum_Clique_Searcher(T_0, n), can be applied to search the solution of the maximum clique problem.

Proof. The algorithm, Maximum_Clique_

Searcher(T_0, n), is implemented via extract, detect and merge operations. Step 1 is the outer loop and is mainly used to form the n tubes and the DNA strands in tube T_i contains i “1”, for $0 \leq i \leq S_1$. Step 2 is an inner loop. On the first execution of step 2a, it uses the extract operation to form two test tubes: T_{j+1}^{on} and T_{j+1}^{off} . Tube T_{j+1}^{on} includes all of the strands that have $x_{i+1} = 1$. Tube T_{j+1}^{off} consists all of the strands that have $x_{i+1} = 0$. On the execution of step 2b, the merge operation is used to pour two tubes T_{j+1}^{on} and T_{j+1}^{off} into tube T_{j+1} . Currently, there will be $j+1$ vertexes appear in the strands of tube T_{j+1} . Repeat execution of step 2a and step 2b until every vertex in the graph are considered. Step 3 is an inner loop, and on the execution of it we will find the maximum clique of the graph. Step 3a is a detect operations to check if tube T_k contains DNA strands. If it returns a ‘yes’, step 3b will be run. On the execution of the inner loop, step 3, the maximum clique will be found. \square

From Maximum_Clique_Searcher(T_0, n), it takes $n \times (n+1)/2$ extract operations, $n \times (n+1)/2$ merge operations, n detect operations, one read operations and n test tubes.

3.6 Improved DNA Algorithm for Maximum Clique Problem

The following DNA algorithm is applied to solve the Maximum Clique Problem.

Algorithm 1. Solving the maximum clique problem.

1. Degree_Searcher(E, V, n).
2. Init_Resolvent_Solution(T_0, k).
3. Parallel_Searcher(T_0, k, E').
4. Clique_Generator($T_0, n-k, E$).
5. Parallel_Searcher($T_0, n-k, E'$).
6. Maximum_Clique_Searcher(T_0, n).

Theorem 1. From those steps in Algorithm 1, the improved DNA based algorithm for maximum clique problem can be solved.

Proof. On the execution of Step 1, it calls Degree_Searcher(E, V, n). The algorithm, Degree_Searcher(E, V, n), is mainly used to find the vertexes whose degrees are “0” or “ $n-1$ ”. The vertexes whose degrees are “0” is sure not to be in the solution of the maximum clique problem, and the vertexes whose degrees are “ $n-1$ ” are sure to be in the maximum clique problem. The algorithm, Init_Resolvent_Solution($T_{02}, n/2$), is mainly used to construct the resolvent solution space for 2^k possible arrays. Step 3 calls Parallel_Searcher(T_0, k, E'). The algorithm, Parallel_Searcher(T_0, k, E')

is employed to search the satisfiable solution from the resolvent solution space. The algorithm, Clique_Generator($T_0, n-k, E$) is mainly used to produce the solution space for the maximum clique problem. With the algorithm, Parallel_Searcher($T_0, n-k, E'$), the illegal solution strands are eliminating from the solution strands and we gain the solution space of the clique problem. Step 6 is called Maximum_Clique_Searcher(T_0, n). The algorithm, Maximum_Clique_Searcher(T_0, n) is mainly used to search the legal DNA strands for maximum clique problem from the clique problem’s solution space. \square

3.7 Performance Analysis of the Proposed DNA Algorithm and Comparison

The following theorems describe time complexity of Algorithm 1, volume complexity of solution space in Algorithm 1, the number of the tube used in Algorithm 1 and the longest library strand in solution space in Algorithm 1.

Theorem 2. The maximum clique problem for any undirected n -vertex graph G with m edges can be solved with $O(n^2)$ biological operations, $O(\sqrt{3}^n)$ strands, $O(n)$ tubes and the longest library strand, $O(n)$ in the Chang et al.’s model, where n is the number of vertices in G and m is at most equal to $(n \times (n-1)/2)$.

Proof. Algorithm 1 includes six main steps. From the algorithm, Step 2, it is very obvious that it takes k “amplify” operations, $2 \times k$ append operations, k merge operations. Step 4 is mainly applied to produce the solution space for the maximum clique problem. It is indicated that it takes $(n-k-1)$ extract operations, $(n-k-1)$ amplify operations, $2 \times (n-k-1)$ append operations, $(n-k-1)$ merge operations. Step 6 is mainly applied to figure out the maximum clique and it takes $n \times (n+1)/2$ extract operations, $n \times (n+1)/2$ merge operations, n detect operations, one read operations and n test tubes.

Step 3 and step 5 is used to eliminate the illegal strands from the solution space. The searcher can be chosen due to the graph’s character.

(1) $m < n \log n$

The graph G is a sparse graph^[14], the sparse parallel searcher is chosen to figure out the solution. The searching will takes $2 \times p$ extract operations ($0 < p < m$), $2 \times p$ merge operations, two detect operations. The number of operations in Algorithm 1 is $n^2 + 7n + 4m - k - 2$.

(2) $m \geq n \log n$

The graph G is a dense graph^[14], the dense

parallel searcher is chosen to figure out the solution. The searching will takes $(n_0 - 1) \times (n_0 - 1)/2$ extract operations, $(n_0 - 1)$ merge operations, $(n_0 - 1)$ detect operations and $(n_0 - 1) \times (n_0 - 1)/2$ discard operations, where $(n_0 - 1) \times (n_0 - 1)/2$ is equal to k or $n - k$ and k is an constant. The number of operations in Algorithm 1 is $2n^2 + 7n - k - 5$.

It is obvious that $O(m) = O(n^2)$. Hence, from the statements mentioned above, it is at once inferred that the time complexity of Algorithm 1 is $O(n^2)$.

After Step 2 of Algorithm1, the resolvent solution is constructed and there are 2^k strands in it. Suppose that there are $M(0 \leq M \leq 2^k)$ strands after the execution of Step 3 where M is constant. Fig. 1 shows the worse complement graph which we can get from the graph G . After the execution of Step 4, the number of strands is less than $M \times \sqrt{3^n}$. Hence, the maximum clique problem can be solved with $O(\sqrt{3^n})$ strands on the Algorithm 1.

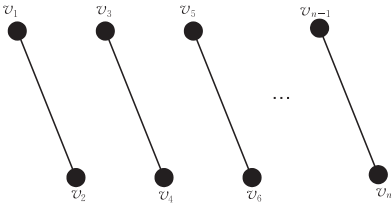


Fig. 1 Complement of a graph G

Refer to the Michael’s algorithm^[11-12], the number of the tube used in Algorithm 1 is $O(n)$. According to Lemma 1 and lemma 2, the longest strand in Algorithm 1 is $O(n)$.

We summarize the resources used by each algorithm for the maximum clique problem in Table 1.

Table 1 Asymptotic Bounds for Resources Required by the Three DNA Algorithm

Algorithm	Solution space size	Length of computation	Number of operations	Number of test tubes
Michael’s ^[11]	$O(2^n)$	$O(n)$	$O(n^2)$	$O(1)$
Quyang’s ^[13]	$O(2^n)$	$O(n)$	$O(n^2)$	$O(1)$
This paper’s	$O(\sqrt{3^n})$	$O(n)$	$O(n^2)$	$O(n)$

As Table 1 showing, our improved algorithm reduces the solution space size of the maximum clique problem with not varying the length of computation and number of operations. But the number of test tubes increases from $O(1)$ to $O(n)$. \square

The Li’s algorithm^[15] for the maximum clique problem is the primary result of the extensible algorithm. It has reduced the solution space size with not varying the length of computation. But for a graph with great vertexes, it can not solve the maximum problem efficiency and the number of

iterations is very big. Our improved algorithm is highly space-efficient and error-tolerant compared to the algorithms for maximum clique problem above.

4 Experimental Results by Simulated DNA Computing

The graph in Fig. 2 denotes such a problem. In Fig. 2, the graph G contains seven vertexes and eleven edges.

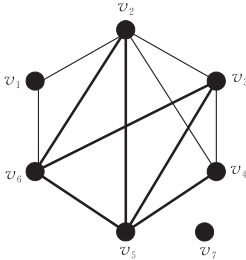


Fig. 2 The graph G of our problem

4.1 DNA Code

Our way to simulate the new algorithm is refer to Michael and Chang at 2004^[11-12].

DNA sequences generated by the modified Adleman program are shown in Table 2. With the nearest neighbor parameters, the Adleman program was used to calculate the enthalpy, entropy, and free energy for the binding of each probe to its corresponding region on a library strand. The energys used are shown in Table 3. The program also figured out the average and standard deviation for the enthalpy, entropy and free energy over all probe/library strand interactions. The energy levels are shown in Table 4.

Table 2 Sequences Used to Represent the 14bits (Blocks) in T_0

Bit	5'→3' DNA sequence	Bit	5'→3' DNA sequence
x_1^0	AATAATTTCTCCCC	x_1^1	CTATCTAATTAACA
x_2^0	ACCTAACCACACTCT	x_2^1	TCCCCACATCAAACC
x_3^0	CTCCTATATCCACCT	x_3^1	CCATAATCACTACCC
x_4^0	TCTACTTTCTCTCAT	x_4^1	ACCCCAACCATCTCA
x_5^0	TTCATAATTAATCCC	x_5^1	TTTAATATTCACATT
x_6^0	CATTCTTTCAATAAC	x_6^1	TCCTACTCATCACA
x_7^0	TTATTACAACCTAAT	x_7^1	CCACAATAAATTTCC

Table 3 The Energys for Binding of Each Probe to Its Corresponding Region on a Library Strand in T_0

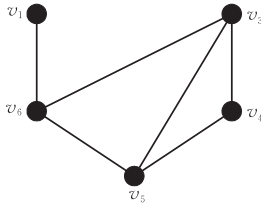
Bit	Enthalpy energy (H)	Entropy energy (S)	Free energy (G)	Bit	Enthalpy energy (H)	Entropy energy (S)	Free energy (G)
x_1^0	114.4	299.4	25.0	x_1^1	107.8	278.6	24.3
x_2^0	103.8	272.6	22.3	x_2^1	112.8	284.9	27.8
x_3^0	115.9	294.0	28.2	x_3^1	111.3	285.7	25.9
x_4^0	108.5	273.0	27.0	x_4^1	105.0	270.8	24.1
x_5^0	111.3	285.7	25.9	x_5^1	105.2	270.5	24.4
x_6^0	108.4	286.3	22.6	x_6^1	109.9	285.5	24.5
x_7^0	111.1	288.3	25.0	x_7^1	101.9	266.0	22.4

Table 4 The Energys over All Probe/Library Strand Interactions in T_0

	Enthalpy energy (H)	Entropy energy (S)	Free energy (G)
Average	109.0929	281.5214	24.95714
Standard deviation	3.935455	9.42786	1.80068

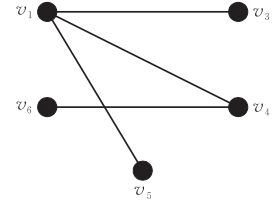
4.2 Solving Process of the Improved Algorithm for the Maximum Clique Problem

After DNA coding, we can simulate the Algorithm 1. First, on the execution of Step 1, finding out the vertex v_2 whose degree is five and the vertex v_7 whose degree is "0". Eliminating the vertex v_2 , the vertex v_7 and all the edges related to the vertex v_2 , we can get a new graph G_1 that shows in Fig. 3. There are five vertexes in graph G_1 and the minimum degree is three.

Fig. 3 The graph G_1

Of the graph G_1 , the vertex set V is $\{v_1, v_3, v_4, v_5, v_6\}$ and the degree set is $\{1, 3, 2, 3, 3\}$. The vertex v_1 having the min degree and the vertex v_6 is adjoin to the vertex v_1 . On the execution of Step 2, the resolvent solution space for the vertex v_1 and v_2 is produced. The library strands are $\{x_1^0 x_6^0, x_1^0 x_6^1, x_1^1 x_6^0, x_1^1 x_6^1\}$. Searching the resolvent solution space, the illegal strands are eliminated and the library strands is $\{x_1^0 x_6^0, x_1^0 x_6^1, x_1^1 x_6^0, x_1^1 x_6^1\}$. Based on the clique generator and the legal relsovent strands, the solution space of the graph G_1 is produced. The library strands are $\{x_1^0 x_6^0 x_3^0 x_4^0 x_5^0, x_1^0 x_6^0 x_3^0 x_4^0 x_5^1, x_1^0 x_6^0 x_3^0 x_4^1 x_5^0, x_1^0 x_6^0 x_3^0 x_4^1 x_5^1, x_1^0 x_6^0 x_3^1 x_4^0 x_5^0, x_1^0 x_6^0 x_3^1 x_4^0 x_5^1, x_1^0 x_6^0 x_3^1 x_4^1 x_5^0, x_1^0 x_6^0 x_3^1 x_4^1 x_5^1, x_1^0 x_6^1 x_3^0 x_4^0 x_5^0, x_1^0 x_6^1 x_3^0 x_4^0 x_5^1, x_1^0 x_6^1 x_3^0 x_4^1 x_5^0, x_1^0 x_6^1 x_3^0 x_4^1 x_5^1, x_1^0 x_6^1 x_3^1 x_4^0 x_5^0, x_1^0 x_6^1 x_3^1 x_4^0 x_5^1, x_1^0 x_6^1 x_3^1 x_4^1 x_5^0, x_1^0 x_6^1 x_3^1 x_4^1 x_5^1, x_1^1 x_6^0 x_3^0 x_4^0 x_5^0, x_1^1 x_6^0 x_3^0 x_4^0 x_5^1, x_1^1 x_6^0 x_3^0 x_4^1 x_5^0, x_1^1 x_6^0 x_3^0 x_4^1 x_5^1, x_1^1 x_6^0 x_3^1 x_4^0 x_5^0, x_1^1 x_6^0 x_3^1 x_4^0 x_5^1, x_1^1 x_6^0 x_3^1 x_4^1 x_5^0, x_1^1 x_6^0 x_3^1 x_4^1 x_5^1, x_1^1 x_6^1 x_3^0 x_4^0 x_5^0, x_1^1 x_6^1 x_3^0 x_4^0 x_5^1, x_1^1 x_6^1 x_3^0 x_4^1 x_5^0, x_1^1 x_6^1 x_3^0 x_4^1 x_5^1, x_1^1 x_6^1 x_3^1 x_4^0 x_5^0, x_1^1 x_6^1 x_3^1 x_4^0 x_5^1, x_1^1 x_6^1 x_3^1 x_4^1 x_5^0, x_1^1 x_6^1 x_3^1 x_4^1 x_5^1\}$.

The graph \bar{G}_1 showing in Fig. 4 has four ledges and five vertexes, so it is a dense graph. We choose the dense parallel searcher for figuring out the satisfied solution. After the execution of Step 5, the library strands are $\{x_1^0 x_6^0 x_3^0 x_4^0 x_5^0, x_1^0 x_6^0 x_3^0 x_4^0 x_5^1, x_1^0 x_6^0 x_3^0 x_4^1 x_5^0, x_1^0 x_6^0 x_3^0 x_4^1 x_5^1, x_1^0 x_6^0 x_3^1 x_4^0 x_5^0, x_1^0 x_6^0 x_3^1 x_4^0 x_5^1, x_1^0 x_6^0 x_3^1 x_4^1 x_5^0, x_1^0 x_6^0 x_3^1 x_4^1 x_5^1, x_1^0 x_6^1 x_3^0 x_4^0 x_5^0, x_1^0 x_6^1 x_3^0 x_4^0 x_5^1, x_1^0 x_6^1 x_3^0 x_4^1 x_5^0, x_1^0 x_6^1 x_3^0 x_4^1 x_5^1, x_1^0 x_6^1 x_3^1 x_4^0 x_5^0, x_1^0 x_6^1 x_3^1 x_4^0 x_5^1, x_1^0 x_6^1 x_3^1 x_4^1 x_5^0, x_1^0 x_6^1 x_3^1 x_4^1 x_5^1, x_1^1 x_6^0 x_3^0 x_4^0 x_5^0, x_1^1 x_6^0 x_3^0 x_4^0 x_5^1, x_1^1 x_6^0 x_3^0 x_4^1 x_5^0, x_1^1 x_6^0 x_3^0 x_4^1 x_5^1, x_1^1 x_6^0 x_3^1 x_4^0 x_5^0, x_1^1 x_6^0 x_3^1 x_4^0 x_5^1, x_1^1 x_6^0 x_3^1 x_4^1 x_5^0, x_1^1 x_6^0 x_3^1 x_4^1 x_5^1, x_1^1 x_6^1 x_3^0 x_4^0 x_5^0, x_1^1 x_6^1 x_3^0 x_4^0 x_5^1, x_1^1 x_6^1 x_3^0 x_4^1 x_5^0, x_1^1 x_6^1 x_3^0 x_4^1 x_5^1, x_1^1 x_6^1 x_3^1 x_4^0 x_5^0, x_1^1 x_6^1 x_3^1 x_4^0 x_5^1, x_1^1 x_6^1 x_3^1 x_4^1 x_5^0, x_1^1 x_6^1 x_3^1 x_4^1 x_5^1\}$. Because of the Step 6, we can get the maximum clique for the clique G_1 . It needs six tubes, T_0, T_1, T_2, T_3, T_4 and T_5 . In tube T_0 , the strand, $\{x_1^0 x_6^0 x_3^0 x_4^0 x_5^0\}$, will appear. The strands, $\{x_1^0 x_6^0 x_3^0 x_4^0 x_5^1, x_1^0 x_6^0 x_3^0 x_4^1 x_5^0, x_1^0 x_6^0 x_3^0 x_4^1 x_5^1, x_1^0 x_6^0 x_3^1 x_4^0 x_5^0, x_1^0 x_6^0 x_3^1 x_4^0 x_5^1, x_1^0 x_6^0 x_3^1 x_4^1 x_5^0, x_1^0 x_6^0 x_3^1 x_4^1 x_5^1, x_1^0 x_6^1 x_3^0 x_4^0 x_5^0, x_1^0 x_6^1 x_3^0 x_4^0 x_5^1, x_1^0 x_6^1 x_3^0 x_4^1 x_5^0, x_1^0 x_6^1 x_3^0 x_4^1 x_5^1, x_1^0 x_6^1 x_3^1 x_4^0 x_5^0, x_1^0 x_6^1 x_3^1 x_4^0 x_5^1, x_1^0 x_6^1 x_3^1 x_4^1 x_5^0, x_1^0 x_6^1 x_3^1 x_4^1 x_5^1, x_1^1 x_6^0 x_3^0 x_4^0 x_5^0, x_1^1 x_6^0 x_3^0 x_4^0 x_5^1, x_1^1 x_6^0 x_3^0 x_4^1 x_5^0, x_1^1 x_6^0 x_3^0 x_4^1 x_5^1, x_1^1 x_6^0 x_3^1 x_4^0 x_5^0, x_1^1 x_6^0 x_3^1 x_4^0 x_5^1, x_1^1 x_6^0 x_3^1 x_4^1 x_5^0, x_1^1 x_6^0 x_3^1 x_4^1 x_5^1, x_1^1 x_6^1 x_3^0 x_4^0 x_5^0, x_1^1 x_6^1 x_3^0 x_4^0 x_5^1, x_1^1 x_6^1 x_3^0 x_4^1 x_5^0, x_1^1 x_6^1 x_3^0 x_4^1 x_5^1, x_1^1 x_6^1 x_3^1 x_4^0 x_5^0, x_1^1 x_6^1 x_3^1 x_4^0 x_5^1, x_1^1 x_6^1 x_3^1 x_4^1 x_5^0, x_1^1 x_6^1 x_3^1 x_4^1 x_5^1\}$, appear in tube T_1 . The strands, $\{x_1^0 x_6^0 x_3^0 x_4^0 x_5^1, x_1^0 x_6^0 x_3^0 x_4^1 x_5^0, x_1^0 x_6^0 x_3^0 x_4^1 x_5^1, x_1^0 x_6^0 x_3^1 x_4^0 x_5^0, x_1^0 x_6^0 x_3^1 x_4^0 x_5^1, x_1^0 x_6^0 x_3^1 x_4^1 x_5^0, x_1^0 x_6^0 x_3^1 x_4^1 x_5^1, x_1^0 x_6^1 x_3^0 x_4^0 x_5^0, x_1^0 x_6^1 x_3^0 x_4^0 x_5^1, x_1^0 x_6^1 x_3^0 x_4^1 x_5^0, x_1^0 x_6^1 x_3^0 x_4^1 x_5^1, x_1^0 x_6^1 x_3^1 x_4^0 x_5^0, x_1^0 x_6^1 x_3^1 x_4^0 x_5^1, x_1^0 x_6^1 x_3^1 x_4^1 x_5^0, x_1^0 x_6^1 x_3^1 x_4^1 x_5^1, x_1^1 x_6^0 x_3^0 x_4^0 x_5^0, x_1^1 x_6^0 x_3^0 x_4^0 x_5^1, x_1^1 x_6^0 x_3^0 x_4^1 x_5^0, x_1^1 x_6^0 x_3^0 x_4^1 x_5^1, x_1^1 x_6^0 x_3^1 x_4^0 x_5^0, x_1^1 x_6^0 x_3^1 x_4^0 x_5^1, x_1^1 x_6^0 x_3^1 x_4^1 x_5^0, x_1^1 x_6^0 x_3^1 x_4^1 x_5^1, x_1^1 x_6^1 x_3^0 x_4^0 x_5^0, x_1^1 x_6^1 x_3^0 x_4^0 x_5^1, x_1^1 x_6^1 x_3^0 x_4^1 x_5^0, x_1^1 x_6^1 x_3^0 x_4^1 x_5^1, x_1^1 x_6^1 x_3^1 x_4^0 x_5^0, x_1^1 x_6^1 x_3^1 x_4^0 x_5^1, x_1^1 x_6^1 x_3^1 x_4^1 x_5^0, x_1^1 x_6^1 x_3^1 x_4^1 x_5^1\}$, appear in tube T_2 . The strand $\{x_1^0 x_6^0 x_3^0 x_4^0 x_5^1, x_1^0 x_6^0 x_3^0 x_4^1 x_5^0, x_1^0 x_6^0 x_3^0 x_4^1 x_5^1, x_1^0 x_6^0 x_3^1 x_4^0 x_5^0, x_1^0 x_6^0 x_3^1 x_4^0 x_5^1, x_1^0 x_6^0 x_3^1 x_4^1 x_5^0, x_1^0 x_6^0 x_3^1 x_4^1 x_5^1, x_1^0 x_6^1 x_3^0 x_4^0 x_5^0, x_1^0 x_6^1 x_3^0 x_4^0 x_5^1, x_1^0 x_6^1 x_3^0 x_4^1 x_5^0, x_1^0 x_6^1 x_3^0 x_4^1 x_5^1, x_1^0 x_6^1 x_3^1 x_4^0 x_5^0, x_1^0 x_6^1 x_3^1 x_4^0 x_5^1, x_1^0 x_6^1 x_3^1 x_4^1 x_5^0, x_1^0 x_6^1 x_3^1 x_4^1 x_5^1, x_1^1 x_6^0 x_3^0 x_4^0 x_5^0, x_1^1 x_6^0 x_3^0 x_4^0 x_5^1, x_1^1 x_6^0 x_3^0 x_4^1 x_5^0, x_1^1 x_6^0 x_3^0 x_4^1 x_5^1, x_1^1 x_6^0 x_3^1 x_4^0 x_5^0, x_1^1 x_6^0 x_3^1 x_4^0 x_5^1, x_1^1 x_6^0 x_3^1 x_4^1 x_5^0, x_1^1 x_6^0 x_3^1 x_4^1 x_5^1, x_1^1 x_6^1 x_3^0 x_4^0 x_5^0, x_1^1 x_6^1 x_3^0 x_4^0 x_5^1, x_1^1 x_6^1 x_3^0 x_4^1 x_5^0, x_1^1 x_6^1 x_3^0 x_4^1 x_5^1, x_1^1 x_6^1 x_3^1 x_4^0 x_5^0, x_1^1 x_6^1 x_3^1 x_4^0 x_5^1, x_1^1 x_6^1 x_3^1 x_4^1 x_5^0, x_1^1 x_6^1 x_3^1 x_4^1 x_5^1\}$, appear in tube T_3 . The tube T_4 and T_5 will be empty. Hence, from the tube T_3 we can find the maximum clique of the graph G_1 which is $\{v_3, v_4, v_5\}$ and $\{v_3, v_5, v_6\}$. Adding the vertex v_2 to it, the maximum clique of the graph G is got, which is $\{v_2, v_3, v_4, v_5\}$ and $\{v_2, v_3, v_5, v_6\}$.

Fig. 4 \bar{G}_1 , the complement of the graph G_1

5 Conclusion

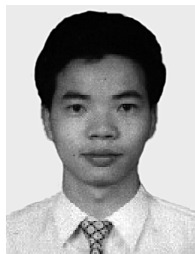
Although there is much preliminary study to the volume's exponential explosion problem^[16-19], it is also the critical factor that constraints the development of the DNA computing. For the objective to decrease the DNA volume of the maximum clique problem, the pruning strategy was taken into the DNA-based supercomputing and a new DNA-based algorithm was proposed. Comparing with the enumerate DNA-based algorithm for maximum clique problem^[11,13-14], the DNA library strands reduced from $O(2^n)$ to $O(\sqrt{3}^n)$.

There are still many NP-complete problems have electronic algorithms where the space complex is lower than $O(2^n)$. Whether these algorithms have corresponding DNA algorithms, or can modify then use in DNA computing? However, currently it is hard to say the molecular computers will have a bright future. In the future molecular computers may be the clear choice for performing massively parallel computations. To reach a free stage in using DNA computers, just as using classical digital computers, there are still many technical difficulties to overcome before this becomes a reality.

References

- [1] Sinden R R. DNA Structure and Function. New York: Academic Press, 1994
- [2] Adleman L M. Molecular computation of solutions to combinatorial problems. Science, 1994, 266: 1021-1024
- [3] Fu B, Beigel R. Length bounded molecular computing. BioSystems, 1999, 52: 155-163

- [4] Bach E, Condon A et al. DNA models and algorithms for NP-complete problems//Proceedings of the 11th Annual Conference on Structure in Complexity Theory. Antwerp, Belgium, 1996; 290-299
- [5] Fu B. Volume bounded molecular computation[Ph. D. dissertation]. Department of Computer Science, Yale University, New Haven, Connecticut, 1997
- [6] Bomze I M, Pelillo M et al. Approximating the maximum weight clique using replicator dynamics. *IEEE Transactions on Neural Networks*, 2000, 11(6): 1228-1241
- [7] Qu Hui-Qin, Lu Ming-Ming et al. Solve partition problem by sticker model in DNA computing. *Progress in Nature Science*, 2004, 14(12): 1116-1121
- [8] Chang W L, Guo M, Michael H. Fast parallel molecular algorithms for DNA-based computation: Factoring integer. *IEEE Transactions on Nanobioscience*, 2005, 4(2): 133-163
- [9] Zimmermann K H. Efficient DNA sticker algorithms for NP-complete graph problems. *Computer Physics Communications*, 2002, 144: 297-309
- [10] Xu J, Li S P et al. Sticker DNA computer model-Part I: Theory. *Chinese Science Bulletin*, 2004, 49(3): 205-212
- [11] Michael H, Chang W L et al. Fast parallel solution for set-packing and clique problems by DNA-based computing. *ICE Transactions on Information and System*, 2004, E87-D(7): 1782-1788
- [12] Michael H. Fast parallel molecular solutions for DNA-based supercomputing: The subsetproduct problem. *BioSystems*, 2005, 80: 233-250
- [13] Quyang Q, Kaplan P D et al. DNA solution of the maximal clique problem. *Science*, 1997, 278: 446-449
- [14] Golumbic M C. *Algorithmic Graph Theory and Perfect Graphs*. New York: Academic Press, 1980
- [15] Li Yuan et al. Genetic algorithm in DNA computing: A solution to the maximal clique problem. *Chinese Science Bulletin*, 2004, 49(9): 967-971(in Chinese)
- [16] Li K L, Yao F J et al. Improved molecular solutions for the knapsack problem on DNA-based supercomputing. *Chinese Journal of Computer Research and Development*, 2007, 44(6): 1063-1070
- [17] Lu S D. *The Experimentation of Molecular Biology*. Beijing: Peking Union Medical College Press, 1999
- [18] Li D F, Li X R et al. Scalability of the surface-based DNA algorithm for 3-SAT. *BioSystems*, 2006, 85: 95-98
- [19] Wang X L, Bao Z M et al. Solving the SAT problem using a DNA computing algorithm based on ligase chain reaction. *BioSystems*, 2008, 91: 17-125



LI Ken-Li, born in 1971, Ph. D., professor, senior member of China Computer Federation. His current main research interests include parallel and distributed processing and molecular computing.

ZHOU Xu, born in 1983, M. S. candidate. Her main research interests is in DNA computing.

ZOU Shu-Ting, born in 1983, M. S. candidate. Her main research interests include DNA computing and Intelligence computing.