PAPER *Special Section on Hardware/Software Support for High Performance Scientific and Engineering Computing*

Fast Parallel Solution for Set-Packing and Clique Problems by DNA-Based Computing

Michael (Shan-Hui) HO[†], **Weng-Long CHANG**[†a)], ***Nonmembers***, **Minyi GUO**[††], ***Member***, ***and*** **Laurence T. YANG**[†††], ***Nonmember***

SUMMARY This paper shows how to use sticker to construct solution space of DNA for the library sequences in the *set-packing problem* and *the clique problem*. Then, with biological operations, we propose DNA-based algorithms to remove illegal solutions and to find legal solutions for the *set-packing* and *clique problems* from the solution space of sticker. Any NP-complete problem in Cook's Theorem can be reduced and solved by the proposed DNA-based computing approach if its size is equal to or less than that of the set-packing problem. Otherwise, Cook's Theorem is incorrect on DNA-based computing and a new DNA algorithm should be developed from the characteristics of the NP-complete problem. Finally, the result to DNA simulation is given.

key words: *biological computing, molecular computing, DNA-based computing*

1. Introduction

Through advances in molecular biology [1]–[4], it is now possible to produce roughly 10^{18} DNA strands that fit in a test tube. Those 10^{18} DNA strands can also be applied for representing 10^{18} bits of information. Basic biological operations can be used to simultaneously operate 10^{18} bits of information. Or we can say that 10^{18} data processors can be executed in parallel.

2. DNA Model of Computation

Distinct nucleotides are detected using their bases, which come from *adenine*, *guanine*, *cytosine* and *thymine*. Those bases are abbreviated as A, G, C and T. Under appropriate conditions two strands of DNA can form a double strand, if the respective bases are the Watson-Crick complements of each other — A matches T and C matches G; also the $3'$ end matches the $5'$ end. The length of a single stranded DNA is the number of nucleotides comprising a single strand. Thus, if a single stranded DNA includes 20 nucleotides, we can say that it is a 20 mer. The length of a double stranded DNA is counted in the number of base pairs. Thus if we make a double stranded DNA from a single stranded 20 mer, then the length of the double stranded DNA is 20 base pairs.

A (test) tube is a set of molecules of DNA (i.e. a multi-set of finite strings over the alphabet $\{A, C, G, T\}$). Given a tube, one can perform the following operations:

1. *Extract*. Given a tube P and a short single strand of DNA called S, we can produce two tubes $+(P, S)$ and $-(P, S)$, where $+(P, S)$ is all of the molecules of DNA in P which consist of the short strand S, and $-(P, S)$ is all of the molecules of DNA in P which do not contain the short strand S.
2. *Merge*. Given tubes P_1 and P_2, yield $\cup(P_1, P_2)$, where $\cup(P_1, P_2) = P_1 \cup P_2$. This operation is to pour two tubes into one, with no change in the individual strands.
3. *Detect*. Given a tube P, if it includes at least one DNA molecule we can say 'yes', and if it contains no DNA we can say 'no'.
4. *Append*. Given a tube P and a short strand of DNA called Z, the operation will append the short strand Z onto the end of every strand in tube P.
5. *Discard*. Given a tube P, the operation will discard the tube P.
6. *Read*. Given a tube P, the operation is used to describe a single molecule, which is contained in tube P. Even if P contains many different molecules, the operation can give an explicit description of only one of them.

3. Using Sticker for Solving the Set-Packing Problem in the Adleman-Lipton Model

3.1 Definition of the Set-Packing Problem

Assume that a finite set S is $\{s_1, \ldots, s_q\}$, where s_m is the mth element for $1 \leq m \leq q$. Also suppose that $|S|$ is the number of elements in S and $|S|$ is equal to q. Assume that C is a collection of subsets to S and C is equal to $\{C_1, \ldots, C_n\}$, where each subset, C_k, is a subset to S for $1 \leq k \leq n$. Also suppose that $|C|$ is the number of subset in C and $|C|$ is equal to n. Suppose that the number of subsets in C is greater than or equal to l, where l is a positive number. Suppose that set-packing for S is a sub-collection $C^1 \subseteq C$ such that C^1 contains at least l mutually distinct sets. In Fig. 1, a finite set S is $\{1, 2\}$ and a collection C is $\{\{1\}, \{2\}\}$ for S. The maximum-size set-packing for S is $\{\{1\}, \{2\}\}$.

$$S = \{1, 2\} \text{ and } C = \{\{1\}, \{2\}\}$$

Fig. 1 The definition of our problem.

Manuscript received October 15, 2003.
Manuscript revised January 15, 2004.

[†] The authors are with the Department of Information Management, Southern Taiwan University of Technology, R.O.C.

[††] The author is with the Department of Computer Software, The University of Aizu, Aizuwakawatsu-shi, 965–8580 Japan.

[†††] The author is with the Department of Computer Science, St. Francis Xavier University, Canada.

a) E-mail: changwl@mail.stut.edu.tw

3.2 Using Sticker for Constructing Solution Space of DNA Sequence for the Set-Packing Problem

Assume that an n-digit binary number corresponds to each possible set-packing to any n-subset collection, C. Also suppose that C^1 is a set-packing for C. If the ith bit in an n-digit binary number is set to 1, then it represents that the ith subset is in C^1. If the ith bit in an n-digit binary number is set to 0, then it represents that the corresponding subset is out of C^1.

Assume that q *one*-digit binary numbers represent q elements in S. Also suppose that the mth one-digit binary number corresponds to the mth element in S. If the kth bit in n bits is set to 1 and the mth element in C_k is distinct from other elements in other subsets, then it represents the corresponding subset in C^1 and the mth element in C_k is included in C^1. Therefore, it is obvious that the mth one-digit binary number is appended onto the tail of those binary numbers, containing the value 1 of the kth bit. If the kth bit in n bits is set to 0, then it represents that every element in C_k is excluded from C^1. This is to say that those elements in C_k are not appended onto the tail of those binary numbers, containing the value 0 of the kth bit.

To implement this, assume that an n-bit binary number Z is represented by a binary number $z_1, \ldots, z_n$, where the value of z_k is 1 or 0 for $1 \leq k \leq n$. A bit z_k is the kth bit in an n-bit binary number Z and it represents the kth subset in C. Assume that q one-digit binary numbers are, subsequently, $y_1, \ldots, y_q$. Assume that y_m represents the mth element in S. An n-bit binary number Z includes all possible 2^n choices of subsets. Each choice of subsets corresponds to a possible set-packing. If the value of z_k is set to 1 and every element in the subset C_k is distinct from other elements in other subsets, then the value 1 for every element in C_k is, subsequently, appended onto the tail of those binary numbers, including the value 1 of the kth bit.

To represent all possible set-packing for the set-packing problem, *sticker* [1]–[3] is used to construct solution space for the problem solved. For every bit z_k representing the kth subset in C, two *distinct* 15 base value sequences were designed. One represents the value "0" of z_k and the other represents the value "1" of z_k. For the sake of convenience in our presentation, assume that ${z_k}^1$ denotes the value of z_k to be 1 and ${z_k}^0$ defines the value of z_k to be 0. Similarly, for every bit y_m representing the mth element in S, two *distinct* 15 base value sequences are also designed. One represents the value, 1, for y_m and the other represents the value, 0, to y_m. For the sake of convenience in our presentation, also assume that ${y_m}^1$ denotes the value of y_m to be 1 and ${y_m}^0$ defines the value of y_m to be 0.

3.3 The DNA Algorithm for Solving of the Set-Packing Problem

Algorithm 1: Solving the set-packing problem.

```
(1) Input $T_0$, where tube $T_0$ is to encode all possible $2^n$ choices of subsets in a collection $C = \{C_1, \ldots, C_n\}$ and every element of each subset in the collection $C$ comes from a finite set $S = \{s_1, \ldots, s_q\}$.
(2) For $k = 1$ to $n$
    (2a) $T_{ON} = +(T_0, {z_k}^1)$ and $T_{OFF} = -(T_0, {z_k}^1)$
    (2b) For $m = 1$ to $|C_k|$
    Assume that the $m$th element in $C_k$ is $s_m$ and $y_m$ is used to represent it.
    (2c) $T_{BAD} = +(T_{ON}, {y_m}^1)$ and $T_{ON} = -(T_{ON}, {y_m}^1)$.
    (2d) Discard($T_{BAD}$).
    (2e) Append($T_{ON}, {y_m}^1$).
    End For
    (2f) $T_0 = \cup(T_{ON}, T_{OFF})$.
    End For
(3) For $k = 0$ to $n - 1$
    For $j = k$ down to 0
    (3a) ${T_{j+1}}^{ON} = +(T_j, {z_{k+1}}^1)$ and $T_j = -(T_j, {z_{k+1}}^1)$.
    (3b) $T_{j+1} = \cup(T_{j+1}, {T_{j+1}}^{ON})$.
  EndFor
  EndFor
(4) For $k = n$ down to 1
    (4a) If (detect ($T_k$) = 'yes') then
    (4b) Read ($T_k$) and terminate the algorithm.
    EndIf
    EndFor
```

Theorem 3-1: From those steps in Algorithm 1, the set-packing problem for a q-element set S and an n-subset collection C can be resolved.

Proof: A test tube of DNA strands, that represent all possible 2^n input bit sequences $z_1, \ldots, z_n$, is yielded in Step 1. It is obvious that the test tube contains all possible 2^n choices of set-packing.

According to the definition of set-packing, Step 2 will be at most executed n^*q times for checking which subsets are disjoint. When the first execution of Step 2(a) uses the "extraction" operation to form two test tubes: T_{ON} and T_{OFF}. The first tube T_{ON} includes all of the strands that have $z_k = 1$, that is to say, the first subset in C occurs in tube T_{ON}. The second tube T_{OFF} consists of all of the strands that have $z_k = 0$, this is to say that the first subset in C does not appear in tube T_{OFF}. Step 2(b) is an inner loop and is used to examine whether those elements in the kth subset are different from other elements in other subsets. On the first execution of Step 2(c), it uses the "extraction" operation to form two test tubes: T_{BAD} and T_{ON}. The first tube T_{BAD} includes all of the strands that have $y_m = 1$. That is to say that the first element in the first subset in C has appeared in tube T_{BAD}. The second tube T_{OFF} contains all of the strands that have $y_m = 0$. This means that the first element of the first subset in C does not appear in tube T_{ON}. From the definition of set-packing, tube T_{BAD} includes illegal strands. Hence, Step 2(d) applies the "discard" operation to discard tube T_{BAD}. Step 2(e) employs the "append" operation to append 15-based DNA sequences for representing the value 1 of y_m into the tail of every strand in T_{ON}. Repeat execution of Steps 2(c) to 2(e) until every element in the first subset in C are processed. Then, Step 2(f) uses the "merge" operation to pour two tubes T_{ON} and T_{OFF} into tube T_0. This is to say that tube T_0 currently contains the strands to represent every element in the first subset of C. Similarly, after other subsets are processed, tube T_0 consists of those strands for representing a legal set-packing.

Each time the outer loop in Step 3 is executed; the number of executions for the inner loop is $(k + 1)$ times. On the first execution of the outer loop, the inner loop is only executed one time. Therefore, Steps 3(a) and 3(b) are executed one time. Step 3(a) uses the “extraction” operation to form two test tubes: T_1^{ON} and T_0. The first tube $T_1{}^{ON}$ contains all of the strands that have $z_1 = 1$. The second tube T_0 consists of all of the strands that have $z_1 = 0$. That is to say that the first tube encodes every set-packing with the first subset in C and the second tube represents every set-packing without the first subset in C. Hence, Step 3(b) applies the “merge” operation to pour tube $T_1{}^{ON}$ into tube T_1. After repeating execution of Steps 3(a) and 3(b), it finally produces n *new* tubes. Tube T_k for $n \geq k \geq 1$ then encodes those DNA strands that contain k subsets.

Because the set-packing problem is to find a maximum-size set-packing, tube T_n first is detected using the “detection” operation in Step 4(a). If it returns “yes”, then tube T_n at least contains a maximum-size set-packing. Therefore, Step 4(b) uses the_ “read” operation for describing the ‘sequence’ of a molecule in tube T_n and terminates the algorithm. Otherwise, continue to repeat execution of Step 4(a) until a maximum-size set-packing is detected in the tube.

The set S and the collection C in Fig. 1 can be applied for showing the power of Algorithm 1. In Fig. 1, the finite set S is {1, 2} and the collection C is {{1}, {2}}. Assume that the collection C is $\{C_1, C_2\}$, where C_1 and C_2 are, {1} and {2} respectively. From Step 1 in Algorithm 1, tube T_0 is filled with four library strands using the techniques mentioned in Sect. 3.2, which represents four possible set-packing for S and C. For the first execution of Step 2(a) in Algorithm 1, two tubes are yielded. The first tube, T_{ON}, includes the numbers 1^* (* can be either 1 or 0). The second tube, T_{OFF}, contains the numbers 0^*. That is to say that the first tube, T_{ON}, includes $\{C_1\}$ and $\{C_1,C_2\}$ and the second tube, T_{OFF}, contains Φ and $\{C_2\}$. Then, for the first execution of Step 2(c), it uses the “extraction” operation to form two test tubes: T_{BAD} and T_{ON}. This is to say that the first element, 1, in C_1 has appeared in T_{BAD} and the first element, 1, in C_1 does not appear in T_{ON}. Step 2(d) applies the “discard” operation to discard tube T_{BAD}. After Step 2(e) is executed, tube T_{ON} contains the bit string: $101(y_1{}^1)$ and $111(y_1{}^1)$. Next, after Step 2(f) is executed, tube T_0 includes the bit strings: 00, 01, $101(y_1{}^1)$ and $111(y_1{}^1)$. Similarly, after the second subset, C_2 is processed, tube T_0 consists of the bit strings: 00, $101(y_1{}^1)$, $011(y_2{}^1)$ and $111(y_1{}^1)\ 1\ (y_2{}^1)$.

The first execution of Step 3(a) in Algorithm 1 applies the “extraction” operation to produce two tubes, $T_1{}^{ON}$ and T_0. Tube $T_1{}^{ON}$ contains the bit strings: $101(y_1{}^1)$, and $111(y_1{}^1)\ 1\ (y_2{}^1)$, while tube T_0 includes the bit strings: 00 and $011(y_2{}^1)$. Step 3(b) applies the “merge” operation to pour tube $T_1{}^{ON}$ into tube T_1. Tube T_1 includes the bit strings, $101(y_1{}^1)$, and $111(y_1{}^1)\ 1\ (y_2{}^1)$, which represents two set-packing, $\{C_1\}$ and $\{C_1, C_2\}$. After repeating execution of Steps 3(a) and 3(b), it produces two *new* tubes. The new tube T_k for $2 \geq k \geq 1$ encodes the legal set packing that contains k subsets, while tube T_2 contains the bit strings, $111(y_1{}^1)\ 1\ (y_2{}^1)$. Tube T_1 includes the bit strings: $101(y_1{}^1)$ and $011(y_2{}^1)$.

The set-packing problem is to find a maximum-size set-packing. The first time of Step 4(a) is executed; tube T_2 is first detected using the “detection” operation. This step returns a “yes” for the “detection” operation for tube T_2. Therefore, Step 4(b) applies the “read” operation to describe the ‘sequence’ of a molecule in tube T_2 and terminates the algorithm. The maximum-size set-packing is found to be $\{C_1, C_2\}$.

3.4 The Complexity of Algorithm 1

Theorem 3-2: Suppose that a finite set S is $\{s_1, \ldots, s_q\}$ and a collection C is $\{C_1, \ldots, C_n\}$, where C_k is a subset of elements from S for $1 \leq k \leq n$. The set-packing problem for S and C can be solved with $O(q^*n + n^2)$ biological operations in the Adleman-Lipton model.

Proof: Algorithm 1 can be applied for solving the set-packing problem for S and C. Algorithm 1 includes three main steps. Step 2 is mainly used to remove illegal library strands from all of the 2^n possible library strands. From Algorithm 1, it is obvious that Steps 2(a) through 2(f) use $(n + q^*n)$ “extraction” operations, (q^*n) “discard” operations, (q^*n) “append” operations and n “merge” operations. Step 3 is mainly used to figure out the number of subsets in every legal set-packing. It is indicated from Algorithm 1 that Step 3(a) takes $(n^*(n + 1)/2)$ “extraction” operations and Step 3(b) takes $(n^*(n + 1)/2)$ “merge” operations. Step 4 is employed to find a maximum-size set-packing from legal set-packing. It is pointed out from Algorithm 1 that Step 4(a) at most takes n “detection” operations and Step 4(b) takes one “read” operation. Hence, from the statements above, it is inferred that the time complexity of Algorithm 1 is $O(q^*n + n^2)$ biological operations in the Adleman-Lipton model.

Theorem 3-3: Suppose that a finite set S is $\{s_1, \ldots, s_q\}$ and a collection C is $\{C_1, \ldots, C_n\}$, where C_k is a subset of elements from S for $1 \leq k \leq n$. The set-packing problem for S and C can be solved with $O(2^n)$ library strands in the Adleman-Lipton model.

Theorem 3-4: Suppose that a finite set S is $\{s_1, \ldots, s_q\}$ and a collection C is $\{C_1, \ldots, C_n\}$, where C_k is a subset of elements from S for $1 \leq k \leq n$. The set-packing problem for S and C can be solved with $O(n)$ tubes in the Adleman-Lipton model.

Theorem 3-5: Suppose that a finite set S is $\{s_1, \ldots, s_q\}$ and a collection C is $\{C_1, \ldots, C_n\}$, where C_k is a subset of elements from S for $1 \leq k \leq n$. The set-packing problem for S and C can be solved with the longest library strand, $O(15^*n+15^*q)$, in the Adleman-Lipton model.

3.5 Range of Application to Cook's Theorem on DNA-Based Computing

Cook's Theorem is that if one algorithm for one NP-complete problem is developed, then other problems will be solved by means of reduction to that problem. The following theorem is used to describe the range of application for Cook's Theorem on DNA-based computing.

Theorem 3-6: Assume that any other NP-complete problems can be reduced to the set-packing problem with a polynomial time algorithm in a general electronic computer. If the size of a reduced NP-complete problem is less than or equal to that of the set-packing problem, then Cook's Theorem is correct on DNA-based computing.

Proof: We transform the clique problem to the set-packing problem with a polynomial time algorithm. Assume that a graph $G = (V, E)$, where V is $\{v_1, v_2, \ldots, v_n\}$ and E is $\{(v_p, v_q)| (v_p, v_q)$ is an edge in $G\}$. Assume that $|E|$ is at most $(n^*(n-1))/2$. Suppose that a graph G is any instance for the clique problem. We construct a finite set S and a collection C, where $S = \{s_1, \ldots, s_{|E|}\}$ and $C = \{C_1, \ldots, C_{|V|}\}$ such that S and C have a maximum-size set-packing if and only if G has a maximum-size clique.

Each vertex u_k in V corresponds to a subset C_k in C for $1 \leq k \leq n$. Each edge in E also corresponds to an element s_m in S for $1 \leq m \leq |E|$. Each subset C_k in C for $1 \leq k \leq n$ is $\{s_m|s_m$ represents an edge containing the vertex v_k for $1 \leq m \leq |E|\}$. Setting S and C from G completes the construction of our instance to the set-packing problem. Therefore, the number of elements in S and C are $|E|$ and n respectively. Algorithm 1 is used to determine the set-packing problem for S and C with 2^n DNA strands, $O(|E|^*n + n^2)$ biological operations, $O(n)$ tubes and the longest library strand, $O(15^*n + 15^*|E|)$. That is to say that Algorithm 1 can be applied to solve the clique problem by means of reducing it to the set-packing problem. Hence, it is derived that if the size of a reduced NP-complete problem is less than or equal to that of the set-packing problem, then Cook's Theorem is correct on DNA-based computing.

From Theorem 3-6, if the size of a reduced NP-complete problem is equal to or less than that of the set-packing problem, then Algorithm 1 can be directly used for solving the reduced NP-complete problem. Otherwise, a new DNA algorithm should be developed from the characteristics of the NP-complete problem.

4. Using Sticker for Solving the Clique Problem in the Adleman-Lipton Model

4.1 The DNA Algorithm for Solving the Clique Problem

Suppose that a graph $G = (V, E)$, where V is $\{v_1, v_2, \ldots, v_n\}$ and E is $\{(v_p, v_q)|(v_p, v_q)$ is an edge in $G\}$. Also Assume that $|V|$ is n and $|E|$ is at most $(n^*(n-1))/2$. Assume that a *complementary* graph $G^1 = (V, E^1)$ for G, where E^1 is $\{(v_c, v_d)|(v_c, v_d)$ is out of $E\}$. Assume that $|E^1|$ is $(n^*(n-1))/2 - |E|$. A *clique* for a graph $G = (V, E)$ is a *complete* sub-graph to G. The clique problem is to find a maximum-size clique in G. Algorithm 2 is presented for solving the clique problem and the notations in Algorithm 2 are similar to those described in Sect. 3.2.

Algorithm 2: Solving the clique problem.

```
(1) Input ($T_0$), where tube $T_0$ encodes all possible $2^n$ choices of clique for a graph $G = (V, E)$, where $|V|$ is $n$ and $|E|$ is at most $(n^*(n-1))/2$.
(2) For $m = 1$ to $(n^*(n-1))/2 - |E|$.
Assume that $e_m$ is an edge in $G^1$ and $e_m = (v_c, v_d)$.
Also suppose that $z_c$ and $z_d$, respectively, represent $v_c$ and $v_d$.
(2a) $T_{ON} = +(T_0, z_c^1)$ and $T_{OFF} = -(T_0, z_c^1)$.
(2b) $T_{ON}^1 = +(T_{ON}, z_d^1)$ and ${T_{OFF}}^1 = -(T_{ON}, {z_d}^1)$.
(2c) Discard(${T_{ON}}^1$).
(2d) $T_0 = \cup(T_{OFF}, {T_{OFF}}^1)$.
EndFor
(3) For $k = 0$ to $n - 1$
    For $j = k$ down to 0
        (3a) ${T_{j+1}}^{ON} = +(T_j, {z_{k+1}}^1)$ and $T_j = -(T_j, {z_{k+1}}^1)$.
        (3b) $T_{j+1} = \cup(T_{j+1}, {T_{j+1}}^{ON})$.
    EndFor
  EndFor
(4) For $k = n$ down to 1
        (4a) If (detect ($T_k$) = 'yes') then
        (4b) Read ($T_k$) and terminate the algorithm.
        EndIf
        EndFor
```

Theorem 4-1: From those steps in Algorithm 2, the clique problem for a graph $G = (V, E)$ can be solved, where $|V|$ is n and $|E|$ is at most $(n^*(n-1))/2$.

Proof: A test tube of DNA strands, that represent all possible 2^n input bit sequences $z_1, \ldots, z_n$, is yielded in Step 1. It is obvious that the test tube contains all possible 2^n choices of clique.

From the definition of clique, Step 2 will be executed $((n^*(n-1))/2 - |E|)$ times for checking which strands are legal. When the first execution of Step 2(a) uses the "extraction" operation to form two test tubes: T_{ON} and T_{OFF}. The first tube T_{ON} includes all of the strands that have $z_c = 1$. That is to say that the cth vertex in V occurs in tube T_{ON}. The second tube T_{OFF} consists of all of the strands that have $z_c = 0$. This is to say that the cth vertex in V does not appear in tube T_{OFF}. Then, at the first execution of Step 2(b), it uses the "extraction" operation to form two test tubes: ${T_{ON}}^1$ and ${T_{OFF}}^1$. The first tube ${T_{ON}}^1$ includes all of the strands that have $z_c = 1$ and $z_d = 1$. That is to say that tube ${T_{ON}}^1$ includes vertices, which are not connected in G. The second tube ${T_{OFF}}^1$ contains all of the strands that have $z_c = 1$ and $z_d = 0$. This is to say that the cth vertex in V occurs in ${T_{OFF}}^1$ and the dth vertex in V does not appear in ${T_{OFF}}^1$. From the definition of clique, tube ${T_{ON}}^1$ includes illegal strands. Hence, Step 2(c) applies the "discard" operation to discard tube ${T_{ON}}^1$. Next, Step 2(d) uses the "merge" operation to pour the two tubes T_{OFF} and ${T_{OFF}}^1$ into tube T_0. This is to say that tube T_0 currently contains legal solutions. After the remaining steps are processed, tube T_0 consists of those strands for representing legal clique.

Each time the outer loop in Step 3 is executed; the number of executions for the inner loop is $(k + 1)$ times. On the first execution of the outer loop, the inner loop is only executed one time. Therefore, Steps 3(a) and 3(b) are executed once. Step 3(a) uses the "extraction" operation to form two test tubes: ${T_1}^{ON}$ and T_0. The first tube ${T_1}^{ON}$ contains all of the strands that have $z_1 = 1$. The second tube T_0 consists of all of the strands that have $z_1 = 0$. That is to say that the first tube encodes every clique with the first vertex in G and the second tube represents every clique without the first vertex in G. Then, Step 3(b) applies the "merge" operation to pour tube ${T_1}^{ON}$ into tube T_1. After repeating execution of Steps 3(a) and 3(b), it finally produces n *new* tubes. The tube T_k for $n \geq k \geq 1$ encodes those DNA strands that contain k vertices.

Because the clique problem is to find a maximum-size clique, tube T_n first is detected with the "detection" operation in Step 4(a). If it returns a "yes", then tube T_n at least contains a maximum-size clique. Therefore, Step 4(b) uses the "read" operation for describing 'sequence' of a molecule in tube T_n and terminating the algorithm. Otherwise, continue to repeat execution Step 4(a) until a maximum-size clique is found in the tube detected.

4.2 The Complexity of Algorithm 2

Theorem 4-2: Suppose that a graph $G = (V, E)$, where $|V|$ is n and $|E|$ is at most $(n^*(n - 1))/2$. The clique problem for G can be solved with $O(n^2 - |E|)$ biological operations in the Adleman-Lipton model.

Proof: Algorithm 2 can be applied for solving the clique problem for G. Algorithm 2 includes three main steps. Step 2 is mainly used to remove illegal library strands from all of the 2^n possible library strands. From Algorithm 2, Steps 2(a) through 2(d) take $(2^*((n^*(n - 1))/2 - |E|))$ for the "extraction" operations, $(n^*(n - 1))/2 - |E|)$ for the "discard" operations and $(n^*(n - 1))/2 - |E|)$ for the "merge" operations. Step 3 is mainly used to figure out the number of vertices in every legal clique. It is indicated from Algorithm 2 that Step 3(a) takes $(n^*(n + 1)/2)$ "extraction" operations and Step 3(b) takes $(n^*(n + 1)/2)$ "merge" operations. Step 4 is employed to find a maximum-size clique from legal clique. It is pointed out from Algorithm 2 that Step 4(a) at most takes n "detection" operations and Step 4(b) takes one "read" operation. Hence, from the statements mentioned above, it is inferred that the time complexity of the worst-case for Algorithm 2 is $O(n^2 - |E|)$ biological operations in the Adleman-Lipton model.

Theorem 4-3: Suppose that a graph $G = (V, E)$, where $|V|$ is n and $|E|$ is at most $(n^*(n - 1))/2$. The clique problem for G can be solved with $O(2^n)$ library strands in the Adleman-Lipton model.

Theorem 4-4: Suppose that a graph $G = (V, E)$, where $|V|$ is n and $|E|$ is at most $(n^*(n - 1))/2$. The clique problem for G can be solved with $O(n)$ tubes in the Adleman-Lipton model.

Theorem 4-5: Suppose that a graph $G = (V, E)$, where $|V|$ is n and $|E|$ is at most $(n^*(n - 1))/2$. The clique problem for G can be solved with the longest library strand, $O(15^*n)$, in the Adleman-Lipton model.

4.3 The Comparison of Algorithm 2 with Algorithm 1 for Dealing with the Clique Problem

From Theorem 3-6, the clique problem for graph $G = (V, E)$ can be directly reduced to the set-packing problem for a finite set S and a collection C. Algorithm 1 can be directly applied to the clique problem for graph $G = (V, E)$ with 2^n DNA strands, $O(|E|^*n+n^2)$ biological operations, $O(n)$ tubes and the longest library strand, $O(15^*n + 15^*|E|)$. From the characteristics of the clique problem for graph $G = (V, E)$, Algorithm 2 is proposed to deal with the same problem of 2^n library strands, $O(n^2 - |E|)$ biological operations, $O(n)$ tubes and the longest library strand, $O(15^*n)$. From the statements above, solving the clique problem, graph $G = (V, E)$, Algorithm 2 is better than Algorithm 1 in the form of biological operations and the longest library strand. This is to imply that an NP-complete problem should correspond to a *new* DNA algorithm.

5. Experimental Results of Simulated DNA Computing

For the convenience of the presentation, simulation of Algorithm 1 is described below in detail. Simulation of Algorithm 2 is similar to that of Algorithm 1. The Adleman program was used to construct each 15-base DNA sequence for every bit of the library. Consider the finite set S and the collection C in Fig. 1. DNA sequences generated by the modified Adleman program are shown in Table 1. The program took one mutation to make new DNA sequences for {1}, {2}, 1 and 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 energy used was shown in Table 2.

These library strands are shown in Table 3 and represent every possible set-packing: ∅, {{2}}, {{1}} and {{1}, {2}}. The program also figured out the average and standard deviation for the enthalpy, entropy and free energy over

Table 1 Sequences chosen were used to represent the two subsets in C and every element of S in Fig. 1.

Bit	5'→ 3' DNA Sequence
${z_1}^0$	AATTCACAAACAATT
${z_2}^0$	ACTCCTTCCCTACTC
${z_1}^1$	TCTCCCTATTTATTT
${z_2}^1$	TCACCAAACCTAAAA
${y_1}^0$	TCTCTCTCTAATCAT
${y_2}^0$	ACTCACATACACCAC
${y_1}^1$	CCATCATCTACCTTA
${y_2}^1$	CAACCTATTATCTTA

Table 2 The energy for binding of each probe to its corresponding region on a library strand.

Bit	Enthalpy energy (H)	Entropy energy (S)	Free energy (G)
z_1^1	114.4	299.4	25
z_1^0	107.8	278.6	24.3
z_2^1	111.5	284.8	26.2
z_2^0	109.1	279	25.9
y_1^1	105.2	270.5	24.4
y_1^0	97.3	252.3	22.1
y_2^1	107	282.4	22.5
y_2^0	94.7	239.8	22.8

Table 3 DNA sequences chosen were used to represent every possible set-packing in tube T_0.

∅	5'-AATTCACAAACAATTACTCCTTCCCTACTC-3' 3'-TTAAGTGTTTGTTAATGAGGAAGGGATGAG-5'
{{2}}	5'-AATTCACAAACAATTTCACCAAACCTAAAA-3' 3'-TTAAGTGTTTGTTAAAGTGGTTTGGATTTT-5'
{{1}}	5'-TCTCCCTATTTATTTACTCCTTCCCTACTC-3' 3'-AGAGGGATAAATAAATGAGGAAGGGATGAG-5'
{{1}{2}}	5'-TCTCCCTATTTATTTTCACCAAACCTAAAA-3' 3'-AGAGGGATAAATAAAAGTGGTTTGGATTTT-5'

Table 4 The energy over all probe/library strand interactions.

	Enthalpy energy (H)	Entropy energy (S)	Free energy (G)
Average	105.875	273.35	24.15
Standard deviation	6.31031	17.7764	1.45002

all probe/library strand interactions. The energy levels are shown in Table 4.

The Adleman program was employed for computing the distribution of the different types of potential mishybridizations. The distribution of the types of potential mishybridizations is the absolute frequency of a probe-strand match of length k from 0 to the bit length 15 (for DNA sequences) where probes are not supposed to match the strands. The distribution was, subsequently, 120, 209, 265, 378, 418, 396, 279, 147, 78, 32, 16, 0, 0, 0, 0 and 0. It is pointed out from the last five zeros that there are 0 occurrences where a probe matches a strand at 11, 12, 13, 14 or 15 places. This shows that the third constraint in subsection 3.2 has been satisfied. Hence, the number of matches peaks at 4(418). That is to say that there are 418 occurrences where a probe matches a strand at 4 places. The results for simulation of Steps 2 through 4 are shown in Tables 5, 6 and 7. From tube T_2, the answer was found to be {{1}, {2}}.

Table 5 DNA sequences generated by Step 2 were applied to represent legal solutions in tube T_0.

Bit string to solution	5'→ 3' DNA Sequence
00	5'-AATTCACAAACAATTACTCCTTCCCTACTC-3'
101(y_1^1)	5'-TCTCCCTATTTATTTACTCCTTCCCTACTCCCATCATCTACCTTA-3'
011(y_2^1)	5'-AATTCACAAACAATTTCACCAAACCTAAAACAACCTATTATCTTA-3'

111(y_1^1)
1(y_2^1)
	5'-TCTCCCTATTTATTTTCACCAAACCTAAAACCATCATCTACCTTACAACCTATTATCTTA-3'

Table 6 DNA sequences generated by Step 3 were employed to represent legal solutions in tubes T_0, T_1 and T_2.

Tube	Bit string to solution	5'→ 3' DNA Sequence
T_0	00	5'-AATTCACAAACAATTACTCCTTCCCTACTC-3'
T_1	101(y_1^1)	5'-TCTCCCTATTTATTTACTCCTTCCCTACTCCCATCATCTACCTTA-3'
	011(y_2^1)	5'-AATTCACAAACAATTTCACCAAACCTAAAACAACCTATTATCTTA-3'

T_2	111(y_1^1)
	1(y_2^1)
		5'-TCTCCCTATTTATTTTCACCAAACCTAAAACCATCATCTACCTTACAACCTATTATCTTA-3'

Table 7 The answer was found from Step 4 in tube T_2.

Bit string To solution	5'→3' DNA sequence

111(y_1^1)
1(y_2^1)
	5'-TCTCCCTATTTATTTTCACCAAACCTAAAACCATCATCTACCTTACAACCTATTATCTTA-3'

6. Conclusions

The proposed algorithms (Algorithm 1 and 2) for solving the set-packing and clique problems are based on biological operations in the Adleman-Lipton model and the solution space of stickers in the sticker-based model. The present algorithms have several advantages from the Adleman-Lipton model and the sticker-based model. First, the proposed algorithm actually has a lower rate of errors for hybridization because we modified the Adleman program to generate good DNA sequences for constructing the solution space of stickers to the set-packing and clique problems. Only simple and fast biological operations in the Adleman-Lipton model were employed to solve the problems. Secondly, those 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. Thirdly, in Algorithm 1 for solving the set-packing problem, the number of tubes, the longest length of DNA library strands and the number of DNA library strands, respectively, are O(n), O($15^*n + 15^*q$) and O(2^n). In Algorithm 2 for solving the clique problem, the number of tubes, the longest length of DNA library strands and the number of DNA library strands, respectively, are O(n), O(15^*n) and O(2^n). This implies that the present algorithms can be easily performed in a fully automated manner in a lab. Fourthly, from the statements in Sect. 4.3, for solving the clique problem, Algorithm 2 is better than Algorithm 1 in the form of biological operations and the longest library strand. Therefore, this seems to imply that Cook's Theorem is unsuitable on a DNA-based computer and a NP-complete problem should correspond to a *new* DNA algorithm.

Currently, there are lots of NP-complete problems that cannot be solved because it is very difficult to support basic biological operations using mathematical operations. We are not sure whether molecular computing can be applied to dealing with every NP-complete problem. Therefore, in the future, our main work is to solve other NP-complete problems that were unresolved with the Adleman-Lipton model and the sticker model.

Acknowledgments

This work is partially supported by National Science Foundation in Taiwan in R.O.C. under the research grants NSC-92-2213-E-218-026-.

References

[1] W.-L. Chang and M. Guo, "Solving the set-cover problem and the problem of exact cover by 3-sets in the Adleman-Lipton's model," BioSystems, vol.72, no.3, pp.263–275, 2003.

[2] W.-L. Chang, M. Ho, and M. Guo, "Molecular solutions for the subset-sum problem on DNA-based supercomputing," BioSystems, vol.73, no.2, pp.117–130, 2004.

[3] W.-L. Chang, M. Ho, and M. Guo, "Fast parallel molecular algorithms for DNA-based computation: Factoring integers," IEEE Fourth Symposium on Bioinformatics and Bioengineering (BIBE2004), pp.125–133, 2004, Taiwan, Republic of China.

[4] L. Adleman, "Molecular computation of solutions to combinatorial problems," Science, vol.266, pp.1021–1024, Nov. 1994.

Michael Ho is an Associate Professor of Southern Taiwan University of Technology, has 25 years industrial and academic experience in the computing field. He had worked as a Sr. Software Engineer and Project Leader developing B2B/B2C/C2C web and multimedia applications and a Sr. database administrator for SQL clustered servers, Oracle, and DB2 databases including network/systems LAN/WAN system administration to US major corporations and government organizations. Dr. Ho had more than 10 years of college teaching/research experience as an Assistant Professor and Research Analyst at Central Missouri State University, the University of Texas at Austin, and BPC International Institute. He earned a Ph.D. in IS/CS with Management and Accounting minor from the University of Texas at Austin and an M.S. degree from St. Mary's University. His research interests include algorithm and computation theories, software engineer, database and data mining, parallel computing, quantum computing and DNA computing.

Weng-long Chang received Ph.D. degrees in Computer Science and Information Engineering from National Cheng Kung University, Taiwan in 1999. His research interests include molecular computing, and languages and compilers for parallel computing.

Minyi Guo received his Ph.D. degree in information science from University of Tsukuba, Japan in 1998. From 1998 to 2000, Dr. Guo had been a research scientist of NEC Soft, Ltd. Japan. He is currently a professor at the Department of Computer Software, The University of Aizu, Japan. From 2001 to 2003, he was a visiting professor of Georgia State University, USA, Hong Kong Polytechnic University, Hong Kong. Dr. Guo has served as general chair, program committee or organizing committee chair for many international conferences, and delivered more than 20 invited talks in USA, Australia, China, and Japan. He is the editor-in-chief of the Journal of Embedded Systems. He is also in editorial board of International Journal of High Performance Computing and Networking, Journal of Embedded Computing, Journal of Parallel and Distributed Scientific and Engineering Computing, and International Journal of Computer and Applications. Dr. Guo's research interests include parallel and distributed processing, parallelizing compilers, data parallel languages, data mining, molecular computing and software engineering. He is a member of the ACM, IEEE, and IEEE Computer Society. He is listed in Marquis Who's Who in Science and Engineering.

Laurence T. Yang is a professor in computer science at St Francis Xavier University, Canada. His research is mainly on high performance scientific and engineering computations with applications, design and test of embedded systems, wireless and mobile computing, pervasive computing and communications.