Available online at www.sciencedirect.com
SCIENCE DIRECT®
BioSystems 72 (2003) 263–275
BioSystems
www.elsevier.com/locate/biosystems

Solving the set cover problem and the problem of exact cover by 3-sets in the Adleman–Lipton model

Weng-Long Chang [a,1], Minyi Guo [b,*]

[a] *Department of Information Management, Southern Taiwan University of Technology, Tainan, Taiwan 701, ROC*
[b] *Department of Computer Software, The University of Aizu, Aizu-Wakamatsu City, Fukushima 965-8580, Japan*

Received 15 April 2003; received in revised form 23 July 2003; accepted 23 July 2003

Abstract

Adleman wrote the first paper in which it is shown that deoxyribonucleic acid (DNA) strands could be employed towards calculating solutions to an instance of the NP-complete Hamiltonian path problem (HPP). Lipton also demonstrated that Adleman's techniques could be used to solve the NP-complete satisfiability (SAT) problem (the first NP-complete problem). In this paper, it is proved how the DNA operations presented by Adleman and Lipton can be used for developing DNA algorithms to resolving the *set cover problem* and the *problem* of *exact cover by* 3-*sets*.

Keywords: Biological computing; Molecular computing; DNA-based computing; The NP-complete problem

1. Introduction

Producing 10^{18} DNA strands that fit in a test tube is possible through advances in molecular biology (Sinden, 1994). Adleman wrote the first paper in which it is shown that each DNA strand could be applied for solving the NP-complete Hamiltonian path problem (Adleman, 1994). Lipton demonstrated that Adleman's experiment could be used to determine the NP-complete satisfiability (SAT) problem (the first NP-complete problem) (Lipton, 1995).

In this paper, we apply the DNA operations in the Adleman–Lipton model to develop two DNA algorithms. The main result of the two DNA algorithms shows that the *set cover problem* and the *problem* of *exact cover by* 3-*sets* can be solved with biological operations in the Adleman–Lipton model. Furthermore, this work represents obvious evidence for the ability of DNA-based computing to solve the NP-complete problems.

The rest of this paper is organized as follows. In Section 2, the Adleman–Lipton model is introduced in detail and the comparison of the model with other models is also given. Section 3 introduces a DNA algorithm for solving the set cover problem. Section 4 describes another DNA algorithms for solving the problem of exact cover by 3-sets. In Section 5, the experimental result of simulated DNA computing is also given. Conclusions are drawn in Section 6.

2. DNA model of computation

In Section 2.1, the summary of DNA structure and the Adleman–Lipton model is described in detail.

* Corresponding author. Tel.: +81-242-37-2557; fax: +81-242-37-2744.
E-mail addresses: changwl@csie.ncku.edu.tw, MhoInCerritos@yahoo.com (W.-L. Chang), minyi@u-aizu.ac.jp (M. Guo).
[1] Tel.: +886-6-2533131x4300; fax: +886-6-2541621.

0303-2647/$ – see front matter © 2003 Elsevier Ireland Ltd. All rights reserved.
doi:10.1016/S0303-2647(03)00149-7

In Section 2.2, the comparison of the Adleman–Lipton model with other models is also introduced in detail.

2.1. The Adleman–Lipton model

A DNA (deoxyribonucleic acid) is a polymer, which is strung together from monomers called deoxyribonucleotides (Sinden, 1994; Paun et al., 1998). Distinct nucleotides are detected only with their bases. Those bases are, respectively, abbreviated as A, G, C and T. Two strands of DNA can form (under appropriate conditions) a double strand, if the respective bases are the Watson–Crick complements of each other—A matches T and C matches G; also 3′ end matches 5′ end. The length of a single stranded DNA is the number of nucleotides comprising the single strand. Thus, if a single stranded DNA includes 20 nucleotides, it is called a 20 mer. The length of a double stranded DNA (where each nucleotide is base paired) 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, also written as 20 bp (for more discussion of the relevant biological background, refer to Sinden, 1994; Boneh et al., 1996; Paun et al., 1998).

The DNA operations proposed by Adleman and Lipton (Adleman, 1994, 1996; Lipton, 1995; Boneh et al., 1996), are described below. These operations will be used for figuring out solutions of the set cover problem and the problem of exact cover by 3-sets in this paper.

The Adleman–Lipton model: 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, S, produce two tubes $+(P, S)$ and $-(P, S)$, where $+(P, S)$ is all of the molecules of DNA in P which contain the strand S as a substrand and $-(P, S)$ is all of the molecules of DNA in P which do not contain the short strand S.
2. *Separate*. Given a tube P and length L for a double strand of DNA, generate one tube $*(P, L)$, where $*(P, L)$ is all of the molecules of DNA in P which length is equal to L.
3. *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 of the individual strands.
4. *Anneal*. The operation is to represent all of the operations that combine a test tube of single stranded DNA with other prepared strands and let them anneal together to form double strands.
5. *Detect*. Given a tube P, say 'yes' if P includes at least one DNA molecule, and say 'no' if it contains none.
6. *Append*. Given a tube P and a short strand of DNA, Z, the operation will append the short strand, Z, onto the end of every strand in the tube P.
7. *Discard*. Given a tube P, the operation will discard the tube P.
8. *Read*. Given a tube P, the operation is used to describe a single molecule, which is contained in the tube P. Even if P contains many different molecules each encoding a different set of bases, the operation can give an explicit description of exactly one of them.

2.2. Other related work and comparison with the Adleman–Lipton model

Based on solution space of *splint* in the Adleman–Lipton model, their methods (Narayanan and Zorbala, 1998; Chang and Guo, 2002a,b,c; Chang et al., 2002) could be applied towards solving the traveling salesman problem, the dominating-set problem, the vertex cover problem, the clique problem, the independent-set problem, the three-dimensional matching problem and the set-packing problem. Those methods for the problems resolved have exponentially increasing volumes of DNA and linearly increasing time.

Bach et al. (1996) proposed an $n1.89^n$ volume, $O(n^2 + m^2)$ time molecular algorithm for the 3-coloring problem and a 1.51^n volume, $O(n^2m^2)$ 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 (1997) 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 (Fu, 1997; Bach et al., 1996) is lower, constructing those volumes is more difficult and the time complexity to the methods is very higher.

Quyang et al. (1997) showed that restriction enzymes could be used to solve the NP-complete clique problem. The maximum number of vertices that they can process is limited to 27 because the size of the pool with the size of the problem exponentially increases (Quyang et al., 1997). Shin et al. (1999) presented an encoding scheme for decreasing error rate in hybridization. The method (Shin et al., 1999) could be employed towards ascertaining the traveling salesman problem for representing integer and real values with fixed-length codes. Arita et al. (1997) and Morimoto et al. (1999), proposed new molecular experimental techniques and a solid-phase method to finding a Hamiltonian path, respectively. Amos (1997) proposed parallel filtering model for resolving the Hamiltonian path problem, the sub-graph isomorphism problem, the 3-vertex-colourability problem, the clique problem and the independent-set problem. Those methods (Arita et al., 1997; Morimoto et al., 1999; Amos, 1997) have lower error rate in real molecular experiments.

In the literatures (Reif et al., 2000; LaBean et al., 2000, 2003), the methods for DNA-based computing by self-assembly require to use DNA nanostructures, called tiles, that have efficient chemistries, expressive computational power, and convenient input and output (I/O) mechanisms. DNA tiles have very lower error rate in self-assembly. Garzon and Deaton (1999) introduced a review of the most important advances in molecular computing.

Adleman and his co-authors (Roweis et al., 1999) proposed sticker-based model to enhance error rate in hybridization in the Adleman–Lipton model. Their model could be used for determining solutions to an instance of the set cover problem. Perez-Jimenez and Sancho-Caparrini (2001) employed sticker-based model (Roweis et al., 1999) to resolve knapsack problems. In our previous work, Chang and Guo (2003) employed the sticker-based model and the Adleman–Lipton model to dealing with the dominating-set problem for decreasing error rate of hybridization.

3. Using the Adleman–Lipton model to solve the set cover problem

In Section 3.1, the summary of the set-cover problem is described. Applying *splints* to constructing solution space of DNA sequences for the set-cover problem is introduced in Section 3.2. In Section 3.3, one DNA algorithm is proposed to resolving the set-cover problem. In Section 3.4, the complexity of the proposed algorithm is described.

3.1. Definition of the set-cover problem

Assume that a finite set S is $\{s_1, \ldots, s_A\}$, where s_k is one element in S for $1 \leq k \leq A$. We denote that $|S|$, which is equal to A, is the cardinality of S. Suppose that a collection C is $\{C_1, \ldots, C_B\}$, where C_i is a subset in S for $1 \leq i \leq B$. We denote that $|C|$ is the cardinality of C and $|C|$ is equal to B. Assume that m is a positive integer. A *set cover* for S is a sub-collection $C^1 \subseteq C$ with $|C^1| \leq m$ such that every element of S belongs to at least one member of C^1 (Cormen et al., 2003; Garey and Johnson, 1979), where $|C^1|$ is the cardinality of C^1. The set cover problem is to find a *minimum-size* set cover for S. For example, a finite set S is $\{1, 2, 3\}$ and a collection C is $\{\{1\}, \{2\}, \{3\}, \{2, 3\}\}$ for S. The *minimum-size* set cover for S is $\{\{1\}, \{2, 3\}\}$. It is pointed out from (Garey and Johnson, 1979) that finding a minimum-size set cover is a NP-complete problem, so it can be formulated as a "search" problem.

3.2. Using splint for constructing solution space of DNA sequence for the set-cover problem

In the Adleman–Lipton model, their main idea is to first generate solution space of DNA sequences for those problems resolved. Then, basic biological operations are used to remove illegal solution and find legal solution from solution space. Assume that a B-digit binary number represents all possible 2^B choices of subsets in S. Also suppose that A *one*-digit binary numbers represent A elements in S.

If one of B bits is set to 1, then it represents that the corresponding subset appears in C^1. If one of B bits is set to 0, then it represents that the corresponding subset is out of C^1. If the ith bit in B bits is set to 1 and the corresponding ith subset contains the element s_k in S, then the kth one-digit binary number is appended

onto the tail of those binary numbers, containing the value 1 of the ith bit. If the ith bit in B bits is set to 1 but the corresponding ith subset does not contain the element s_k in S, then the kth one-digit binary number is not appended onto the tail of those binary numbers, consisting of the value 1 of the ith bit.

Suppose that C_i and C_j are, respectively, the ith subset and the jth subset in C^1. Assume that C_i contains the element s_k in S but C_j does not include s_k. From the statements above, the ith bit and the jth bit in B bits are both set 1. Because C_i contains the element s_k, the kth one-digit binary number is appended onto the tail of B bits.

To implement this way, assume that a B-bit binary number X is represented as a binary number $x_1, \ldots, x_B$, where the value of x_i is 1 or 0 for $1 \leq i \leq B$. Similarly suppose that A one-digit binary number are, respectively, $y_1, y_2, \ldots, y_A$, where the value of y_k is 1 or 0 for $1 \leq k \leq A$. A B-bit binary number X includes all possible 2^B choices of subsets. Each choice of subsets corresponds to a possible set cover. A bit x_i is the ith bit in X and it represents the ith subset in C. A one-digit binary number y_k represents the element s_k in S. If the value of x_i is set to 1 and the corresponding subset also consists of the element s_k, then the one-digit binary number y_k is appended onto the tail of those binary numbers, including the value 1 of the ith bit.

Consider that a finite set S is $\{1, 2, 3\}$ and a collection C is $\{\{1\}, \{2\}, \{3\}, \{2, 3\}\}$ for S. The *minimum-size* set cover for S is $\{\{1\}, \{2, 3\}\}$. From the implemented way above, the collection C includes four subsets, so an unsigned integer X with length of four bits is used to represent all possible 2^4 choices of subsets. Since the finite set S contains three elements, three one-digit binary numbers, y_1, y_2 and y_3, respectively, represent the first element, the second element and the third element. The minimum-size set cover for S consists of $\{1\}$ and $\{2, 3\}$. The subset $\{1\}$ is the first subset in C and the subset $\{2, 3\}$ is the fourth subset in C. Therefore, a four-digit binary number, 1001, is applied to represent the first subset $\{1\}$ and the fourth subset $\{2, 3\}$. Because the first bit in 1001 is 1 and the corresponding subset also includes the first element 1 in S, the one-digit binary number y_1 is appended onto the tail of 1001. Hence, the solution currently becomes 10011. Similarly, since the fourth bit in 10011 is 1 and the corresponding subset also includes the second element 2 and the third element 3 in S, the corresponding one-digit binary numbers y_2 and y_3 are, in order, appended onto the tail of 10011. Therefore, the complete solution is constructed and it is 1001111.

To represent all of the possible set cover, *splint* (Adleman, 1994, 1996; Lipton, 1995; Boneh et al., 1996) is used to construct solution space for that problem resolved. For the sake of convenience of presentation, assume that x_i^1 denotes the value of x_i to be 1 and x_i^0 defines the value of x_i to be zero. Similarly, suppose that y_k^1 denotes the value of y_k to be 1 and y_k^0 defines the value of y_k to be zero. Two *distinct* 30 base value sequences were designed for x_i^1 and y_k^1. Similarly, two *distinct* 20 base value sequences also were designed for x_i^0 and y_k^0. The first advantage for the design of DNA code allows to separate DNA strands by length. The *shortest* strand corresponds to a minimum-size set cover. The second advantage for the design of DNA code decreases number of tubes used. Splint's technology in (Adleman, 1994, 1996; Lipton, 1995; Boneh et al., 1996) is applied to construct solution space for all of the possible set cover for any A-element set.

3.3. The DNA algorithm for solving the set-cover problem

The following DNA algorithm is proposed to solve the set-cover problem.

Algorithm 1. Solving the set-cover problem.

(1) Input (P), where the tube P is to encode all possible 2^B choices of subsets for a finite set $S = \{s_1, s_2, \ldots, s_A\}$ and a collection $C = \{C_1, \ldots, C_B\}$ of subsets of S.
(2) For $i = 1$ to B, where B is the number of elements in C.
  (2a) $P_{\mathrm{ON}} = +(P, x_i^1)$ and $P_{\mathrm{OFF}} = -(P, x_i^1)$.
  (2b) For $m = 1$ to $|C_i|$, where $|C_i|$ is the number of elements in C_i.
  Assume that the mth element in C_i is the jth element, s_j, in S.
  (2c) $Q^1 = +(P_{\mathrm{ON}}, y_j^1)$ and $Q^2 = -(P_{\mathrm{ON}}, y_j^1)$.
  (2d) Append (Q^2, y_j^1).
  (2e) $P_{\mathrm{ON}} = \cup(Q^1, Q^2)$.
  End For
(2f) $P = \cup(P_{\mathrm{ON}}, P_{\mathrm{OFF}})$.

End For
(3) For $j = 1$ to A
  (3a) $P = +(P, y_j^1)$ and $P_{\mathrm{BAD}} = -(P, y_j^1)$.
  (3b) Discard the tube P_{BAD}.
End For
(4) $P =$ Anneal (P).
(5) $P = *(P, l)$, where l is equal to the shortest length for double strands of DNA in the tube P.
(6) If (detect (P) = '*yes*') then.
  (6a) Read (P), where the operation describes 'sequence' of a molecular in the tube P.
End If

Theorem 1. *From those steps in* Algorithm 1, *the set-cover problem for any A-element set S and any B-subset collection C can be resolved.*

Proof. In Step 1, a test tube of DNA strands, that encode all 2^B possible input bit sequences $x_1 \ldots x_B$, is yielded. It is very obvious that the test tube includes all possible 2^B choices of subsets. Based on the definition of set cover (Cormen et al., 2003; Garey and Johnson, 1979), Step 2 will execute B times for representing every element in each subset. Step 2a uses "extraction" operation to form two test tubes: P_{ON} and P_{OFF}. The first tube P_{ON} consists of all of the strands that have $x_i = 1$. That is to say that the ith subset appears in the tube P_{ON}. The second tube P_{OFF} includes all of the strands that have $x_i = 0$. That is to say that the ith subset does not appear in the tube P_{OFF}. Step 2b will execute $|C_i|$ times for representing all of the elements in the ith subset. Step 2c applies "extraction" operation from the tube P_{ON} to generate two tubes: Q^1 and Q^2. The first tube Q^1 includes all of the strands that have $y_j = 1$. That is to say that the jth element in the ith subset appears in the tube Q^1. The second tube Q^2 contains all of the strands that have $y_j = 0$. That is to say that the jth element in the ith subset does not appear in the tube Q^2. It is pointed out from the definition of set cover that Step 2d appends the short strand, y_j^1, representing the jth element in the ith subset, onto the end of every strand in the tube Q^2. Hence, the tube Q^2 now includes the jth element in the ith subset. Step 2e applies "merge" to pour two tubes Q^1 and Q^2 into the tube P_{ON}. After repeat to execute $|C_i|$ times for Steps 2c to 2e, all of the elements in the ith subset are represented in the tube P_{OFF}. Step 2f uses "merge" to pour two tubes P_{ON} and P_{OFF} into the tube P. Therefore, the tube P contains every element of the ith subset. It is very obvious that after all of the steps in Step 2 are executed, every element in each subset appears in the tube P. Also from the definition of set cover, Step 3 is applied to check which subsets at least contain every element. Because the number of elements is A, Step 3 will execute A times for finding correct choices of subsets. Step 3a applies "extraction" operation to check which subsets include the jth element and which subsets do not include the jth element. Therefore, two tubes are generated. Clearly, the second tube P_{BAD} includes illegal choices of subsets and therefore the tube P_{BAD} is discarded in Step 3b. After repeat to execute A times for Step 3a and Step 3b, all illegal choices of subsets are discarded. Therefore, the remaining strands in the tube P represent legal choices of subsets and they exist in the tube in the form of single stranded DNA. Step 4 is to combine the single strands in the tube P with other prepared strands and let them anneal together to form double strands. Step 5 separates the strands in the tube P by length. The *shortest* strand corresponds to a minimum-size set cover. □

Algorithm 1 can be used for determining a minimum-size set cover to that a finite set S is $\{1, 2, 3\}$ and a collection C is $\{\{1\}, \{2\}, \{3\}, \{2, 3\}\}$ for S. From Step 1 of Algorithm 1, the tube P is filled with 16 double strands of DNA, representing 16 possible choices of subsets. The number of the subsets in the collection C is four, so the number of execution to Step 2 of Algorithm 1 is four times. According to the first execution of Step 2a of Algorithm 1, two tubes are generated. The first tube, P_{ON}, includes the numbers $1***$ ($*$ can be either 1 or 0). The second tube, P_{OFF}, contains the numbers $0***$. That is to say that the first subset, $\{1\}$, appears in P_{ON} and out of P_{OFF}. Because the number of the element in the first subset is one, the number of execution to Step 2b of Algorithm 1 is one time. Due to the first execution of Step 2c of Algorithm 1, two tubes are produced. The first tube, Q^1, contains the element, 1. The second tube, Q^2, does not include the element, 1. From the definition of set cover, the tube, Q^2, must consist of the element, 1. Therefore, Step 2d of Algorithm 1 appends the element, 1, onto the tube Q^2. According to definition of

set cover, Step 2e of Algorithm 1 pours the two tubes, Q^1 and Q^2, into the tube P_{ON}. After finish each operation of Steps 2c to 2e, the only element in the first subset, $\{1\}$, appears in the tube P_{ON}. From the definition of set cover, Step 2f pours the two tubes, P_{ON} and P_{OFF}, into the tube P. This is to say that every element in the first subset is represented in the tube P.

The same processing can be applied to deal with other three subsets: $\{2\}$, $\{3\}$, $\{2, 3\}$. After the three subsets are processed, every element in the three subsets is represented in the tube P. According to definition of set cover, finding a set cover for S is to check whether every element in S appears at least in chosen subsets. Because the number of the elements in S is three, Step 3 of Algorithm 1 will be executed three times. According to the first execution of Step 3a of Algorithm 1, two tubes are produced. The first element in S is 1 and it is included in the first tube P. The second tube P_{BAD} does not contain the first element. This is to imply that subsets in the second tube P_{BAD} are not legal choices. Therefore, Step 3b is applied to discard the second tube P_{BAD}. The similar processing can be applied to deal with other two elements: 2 and 3. After every element is processed, the remaining strands in the tube P represent legal choices. So, finding a minimum-size set cover for S is to search the *shortest* length for the strands in the tube P. Steps 4 and 5 of Algorithm 1 are applied to find the answer from the tube P. Steps 6 and 6a of Algorithm 1 read the answer from the tube P. Thus, a minimum-size set cover for S is $\{\{1\}, \{2, 3\}\}$. The following theorems describe complexity of Algorithm 1.

Theorem 2. *Assume that a finite set S is $\{s_1, s_2, \ldots, s_A\}$ and a collection C is $\{C_1, \ldots, C_B\}$. The set-cover problem for the finite set S and the collection C can be resolved with $O(B * A + A)$ biological operations in the Adleman–Lipton model, where B is the number of subsets in C and A is the number of elements in S.*

Proof. Refer to Theorem 1. □

Theorem 3. *Assume that a finite set S is $\{s_1, s_2, \ldots, s_A\}$ and a collection C is $\{C_1, \ldots, C_B\}$. The set-cover problem for the finite set S and the collection C can be resolved with constant tubes in the Adleman–Lipton model.*

Proof. Refer to Theorem 1. □

Theorem 4. *Assume that a finite set S is $\{s_1, s_2, \ldots, s_A\}$ and a collection C is $\{C_1, \ldots, C_B\}$. The set-cover problem for the finite set S and the collection C can be resolved with 2^B strands in the Adleman–Lipton model.*

Proof. Refer to Theorem 1. □

Theorem 5. *Assume that a finite set S is $\{s_1, s_2, \ldots, s_A\}$ and a collection C is $\{C_1, \ldots, C_B\}$. The set-cover problem for the finite set S and the collection C can be resolved with the longest strand, $30 * (B + A)$ base pairs, in the Adleman–Lipton model, where B is the number of subsets in C and A is the number of elements in S.*

Proof. Refer to Theorem 1. □

4. Using the Adleman–Lipton model to solve the problem of exact cover by 3-sets

Another famous NP-complete problem is 3-set exact cover problem. In this section, we try to use DNA-based computing technique to solve it. The notations, DNA solution space and processing step are similar with Section 3, in which we solved the set cover problem by DNA-based computing. So we directly give Algorithm 2 in this section.

Assume that a finite set S is $\{s_1, s_2, \ldots, s_{3q}\}$, where s_k is one element in S for $1 \leq k \leq 3q$. We denote that $|S|$, which is equal to $3q$, is the cardinality of S. Assume that C is a collection of 3-element subsets to S, and $|C|$ is the cardinality of C. Suppose that C is equal to $\{C_1, \ldots, C_B\}$, where each subset, C_i, contains three elements in S for $1 \leq i \leq B$. An *exact cover* for S is a sub-collection $C^1 \subseteq C$ such that every element of S occurs in exactly one member of C^1 (Cormen et al., 2003; Garey and Johnson, 1979). The problem of exact cover by 3-sets is to find a *minimum-size* exact cover for S and has been proved to be NP-complete problem (Garey and Johnson, 1979). For example, a finite set S is $\{1, 2, 3, 4, 5, 6\}$ and a collection C is $\{\{1, 2, 3\}, \{3, 4, 5\}, \{4, 5, 6\}\}$ for S. The set S and the collection C denote such a problem. The *minimum-size* exact cover for S is $\{\{1, 2, 3\}, \{4, 5, 6\}\}$. We design the following

DNA algorithm towards resolving the problem. The similar notations and the similar processing steps as Algorithm 1 will be used in Algorithm 2.

Algorithm 2. Solving the problem of exact cover by 3-sets.

(1) Input (P), where the tube P is generated by the same method in Section 3.2. The tube P is to encode all possible 2^B choices of subsets for a finite set $S = \{s_1, s_2, \ldots, s_{3q}\}$ and a collection $C = \{C_1, \ldots, C_B\}$ of 3-element subsets of S.
(2) For $i = 1$ to B, where B is the number of elements in C.
  (2a) $P_{\mathrm{ON}} = +(P, x_i^1)$ and $P_{\mathrm{OFF}} = -(P, x_i^1)$.
  (2b) For $m = 1$ to $|C_i|$, where $|C_i|$ is the number of elements in C_i.
  Assume that the mth element in C_i is the jth element, s_j, in S.
  (2c) $P_{\mathrm{BAD}} = +(P_{\mathrm{ON}}, y_j^1)$ and $P_{\mathrm{ON}} = -(P_{\mathrm{ON}}, y_j^1)$.
  (2d) Discard the tube P_{BAD}.
  (2e) Append (P_{ON}, y_j^1).
  End For
  (2f) $P = \cup(P_{\mathrm{ON}}, P_{\mathrm{OFF}})$.
End For
(3) For $j = 1$ to $3q$
  (3a) $P = +(P, y_j^1)$ and $P_{\mathrm{BAD}} = -(P, y_j^1)$.
  (3b) Discard the tube P_{BAD}.
End For
(4) $P =$ Anneal (P).
(5) $P = *(P, l)$, where l is equal to the shortest length for double strands of DNA in the tube P.
(6) If (detect $(P) =$ '*yes*') then.
  (6a) Read (P), where the operation describes 'sequence' of a molecular in the tube P.
  End If

Theorem 6. *From those steps in Algorithm 2, the problem of exact cover by 3-sets for any $3*q$-element set and any B-subset collection C can be resolved.*

Proof. In Step 1, a test tube of DNA strands, that encode all possible 2^B input bit sequences $x_1 \ldots x_B$, is generated. It is very clear that the test tube includes all possible 2^B choices of 3-element subsets. It is pointed out from the definition of exact cover by 3-sets (Garey and Johnson, 1979) that Step 2 will execute B times for representing every element in each subset. Step 2a uses "extraction" operation to form two test tubes: P_{ON} and P_{OFF}. The first tube P_{ON} consists of all of the strands that have $x_i = 1$. That is to say that the ith subset appears in the tube P_{ON}. The second tube P_{OFF} includes all of the strands that have $x_i = 0$. That is to say that the ith subset does not occur in the tube P_{OFF}. From the definition of exact cover by 3-sets, the number of elements in each subset in C is all three. Therefore, Step 2b will execute three times for representing all of the elements in the ith subset. Step 2c applies "extraction" operation to generate two tubes: P_{BAD} and P_{ON}. The first tube P_{BAD} includes all of the strands that have $y_j = 1$. That is to say that the jth element in the ith subset repeatedly occurs in the tube P_{BAD}. From the definition of exact cover by 3-sets (Garey and Johnson, 1979), the first tube P_{BAD} contains illegal choices of subsets. Hence, the first tube P_{BAD} is discarded in Step 2d. The second tube P_{ON} includes all of the strands that have $y_j = 0$. That is to say that the jth element in the ith subset does not appear in the tube P_{ON}. Obviously, from the definition of exact cover by 3-sets (Garey and Johnson, 1979) that Step 2e appends the short strand, y_j^1, representing the jth element in the ith subset, onto the end of every strand in the tube P_{ON}. Hence, the tube P_{ON} now includes the jth element in the ith subset. After repeat to execute three times for Steps 2c to 2e, all of the elements in the ith subset are represented in the tube P_{ON}. Step 2f uses "merge" operation to pour two tubes P_{ON} and P_{OFF} into the tube P. Therefore, the tube P contains every element of the ith subset. It is very clear that after all of the steps in Step 2 are executed, every element in each subset appears once in the tube P. Step 3 is applied to check which subsets exactly contain every element. Because the number of elements in S is $3q$, Step 3 will execute $3q$ times for finding correct choices of subsets. Step 3a applies "extraction" operation to check which subsets include the jth element and which subsets do not include the jth element. Therefore, two tubes are generated. From the definition of exact cover by 3-sets (Garey and Johnson, 1979), the second tube P_{BAD} includes illegal choices of subsets and therefore the tube P_{BAD} is discarded in Step 3b. After repeat to execute $3q$ times for Step

3a and Step 3b, all illegal choices of subsets are discarded. Therefore, the remaining strands in the tube P represent legal choices of subsets and they exist in the tube in the form of single stranded DNA. Step 4 is to combine the single strands in the tube P with other prepared strands and let them anneal together to form double strands. Step 5 separates the strands in the tube P by length. The *shortest* strand corresponds to a minimum-size exact cover. □

Algorithm 2 can be used for determining a minimum-size set cover to that a finite set S is {1, 2, 3, 4, 5, 6} and a collection C is {{1, 2, 3}, {3, 4, 5}, {4, 5, 6}} for S. From Step 1 of Algorithm 2, the tube P is filled with eight double strands of DNA, representing eight possible choices of subsets. The number of the subsets in the collection C is three, so the number of execution to Step 2 of Algorithm 2 is three times. According to the first execution of Step 2a of Algorithm 2, two tubes are generated. The first tube, P_{ON}, includes the numbers $1**$ ($*$ can be either 1 or 0). The second tube, P_{OFF}, contains the numbers $0**$. That is to say that the first subset, {1, 2, 3}, appears in P_{ON} and out of P_{OFF}. Because the number of the element in the first subset is three, the number of execution to Step 2b of Algorithm 2 is three times. Due to the first execution of Step 2c of Algorithm 2, two tubes are produced. The first tube, P_{BAD}, contains the element, 1. According to definition of exact cover by 3-sets (Garey and Johnson, 1979), subsets in the tube P_{BAD} are illegal choices of subsets. Therefore, the tube P_{BAD} is discarded in Step 2d. The second tube, P_{ON}, does not include the element, 1. From the definition of exact set by 3-sets (Garey and Johnson, 1979), the tube, P_{ON}, must consist of the element, 1. Therefore, Step 2e of Algorithm 2 appends the element, 1, onto the tube P_{ON}. After finish each operation of Steps 2c to 2e, the three elements in the first subset, {1, 2, 3}, appears once in the tube P_{ON}. Also From the definition of exact cover by 3-sets (Garey and Johnson, 1979), Step 2f pours the two tubes, P_{ON} and P_{OFF}, into the tube P. This is to say that every element in the first subset is represented in the tube P.

The similar processing can be applied to deal with other two subsets: {3, 4, 5}, {4, 5, 6}. After the two subsets are processed, every element in the two subsets is represented in the tube P. According to definition of exact cover by 3-sets (Garey and Johnson, 1979), finding an exact cover for S is to check whether every element in S appears exactly in chosen subsets. Because the number of the elements in S is six, Step 3 of Algorithm 2 will be executed six times. According to the first execution of Step 3a of Algorithm 2, two tubes are produced. The first element in S is 1 and it is included in the first tube P. The second tube P_{BAD} does not contain the first element. This is to imply that subsets in the second tube P_{BAD} are not legal choices. Therefore, Step 3b are applied to discard the second tube P_{BAD}. The similar processing can be applied to deal with other five elements: 2, 3, 4, 5 and 6. After every element is processed, the remaining strands in the tube P represent legal choices. So, finding a minimum-size exact cover for S is to search the *shortest* length for the strands in the tube P. Steps 4 and 5 of Algorithm 2 are applied to find the answer from the tube P. Steps 6 and 6a of Algorithm 2 read the answer from the tube P. Thus, a minimum-size exact cover for S is {{1, 2, 3}, {4, 5, 6}}. The following theorems describe the complexity of Algorithm 2.

Theorem 7. *Assume that a finite set S is* $\{s_1, s_2, \ldots, s_{3q}\}$ *and a collection C is* $\{C_1, \ldots, C_B\}$*, where each subset in C includes only three elements in S. The problem of exact cover by* 3*-sets for the finite set S can be resolved with* $O(3*B+3*q)$ *biological operations in the Adleman–Lipton model, where B is the number of subsets in C and* $3*q$ *is the number of elements in S.*

Proof. Refer to Theorem 6. □

Theorem 8. *Assume that a finite set S is* $\{s_1, s_2, \ldots, s_{3q}\}$ *and a collection C is* $\{C_1, \ldots, C_B\}$*, where each subset in C includes only three elements in S. The problem of exact cover by* 3*-sets for the finite set S can be resolved with constant tubes in the Adleman–Lipton model.*

Proof. Refer to Theorem 6. □

Theorem 9. *Assume that a finite set S is* $\{s_1, s_2, \ldots, s_{3q}\}$ *and a collection C is* $\{C_1, \ldots, C_B\}$*, where each subset in C includes only three elements in S. The problem of exact cover by* 3*-sets for the finite set S can be resolved with* 2^B *strands in the Adleman–Lipton model, where B is the number of subsets in C.*

Proof. Refer to Theorem 6. □

Theorem 10. *Assume that a finite set S is* $\{s_1, s_2, \ldots, s_{3q}\}$ *and a collection C is* $\{C_1, \ldots, C_B\}$*, where each subset in C includes only three elements in S. The problem of exact cover by* 3*-sets for the finite set S can be resolved with the longest strand,* $30 * (B + 3 * q)$ *base pairs, in the Adleman–Lipton model, where B is the number of subsets in C and* $3 * q$ *is the number of elements in S.*

Proof. Refer to Theorem 6. □

5. Experimental results of simulated DNA computing

We developed a tool for simulating biological operations in the Adleman–Lipton model in Section 2. In simulations, the rule of DNA code design is that of vertex color. In simulations, it is suggested that four bases {A, C, G, T} represent four distinct colors. The same base (color) do not construct two adjacency nucleotides in single-stranded DNA. To avoid common nucleotides in different strands, if a color sequence is applied to produce a new single-stranded DNA, then it could not be used to generate other new strands. In simulations, due to the rule of DNA code design, single strand of DNA is generated for every element in a finite set and every subset in a collection of the finite set. Similarly, from the definition of splint, single stranded DNA to each splint is also yielded. According to the rule of DNA code design, single strand of DNA is yielded for a $3' \rightarrow 5'$ sequence complementary to the first half of the initial bit (its value was zero) and a $3' \rightarrow 5'$ sequence complementary to the first half of the initial bit (its value is 1). Similarly, due to the rule of DNA code design, single strand of DNA is produced to a $3' \rightarrow 5'$ sequence complementary to the last half of the last bit (its value is 0) and a $3' \rightarrow 5'$ sequence complementary to the last half of the last bit (its value is 1).

Consider that a finite set S is $\{1, 2\}$ and a collection C is $\{\{1\}, \{2\}\}$ for the finite set S. The first element and the second element are, respectively, 1 and 2 for S. The first subset and the second subset are, subsequently, $\{1\}$ and $\{2\}$ for C. From the rule above, single stranded DNA for the every subset and every element is shown in Tables 1 and 2, respectively. Similarly, single stranded DNA to each splint is also shown in Table 3. A $3' \rightarrow 5'$ sequence complementary to the first half of the initial bit (x_1^0) and a $3' \rightarrow 5'$ sequence complementary to the first half of the initial bit (x_1^1) are, respectively, CTAGTCGTGA and TAGCTGCGTAGCGAG. Similarly, a $3' \rightarrow 5'$ sequence complementary to the last half of the last bit (x_2^0) and a $3' \rightarrow 5'$ sequence complementary to the last half of the last bit (x_2^1) are TCTATACAGA and GTCGCTCTGCTCTGT, respectively.

Table 1
DNA sequences chosen to represent the subsets in the collection C

Subset	$5' \rightarrow 3'$ DNA sequence
x_1^0	GATCAGCACTGCTAGTCACA
x_1^1	ATCGACGCATCGCTCACGTGTGCGTGCTGC
x_2^0	TAGTGTAGCGAGATATGTCT
x_2^1	GCGATGTCGCTCGTGCAGCGAGACGAGACA

Table 2
DNA sequences chosen to represent the elements in the finite set S

Element	$5' \rightarrow 3'$ DNA sequence
y_1^1	GTATCGATCAGTGATGTGACATAGAGTAGA
y_2^1	ACTGTACGAGTGTCTATAGACTCAGTGCAC

In simulations, many copies of DNA sequences, representing every element and every subset, are generated. Similarly, many copies of a $3' \rightarrow 5'$ sequence complementary to the first half of the initial bit (its value is 0) and many copies of a $3' \rightarrow 5'$ sequence complementary to the first half of the initial bit (its value is 1) are also yielded. Next, in simulations, many copies of a $3' \rightarrow 5'$ sequence complementary to the last half of the last bit (its value is 0) and many copies of a $3' \rightarrow 5'$ sequence complementary to the last half of the last bit (its value is 1) are also produced. From

Table 3
DNA sequences chosen were regarded as the splint for adjacency bits in Table 1

Splint	$3' \rightarrow 5'$ DNA sequence
(x_1^0, x_2^0)	CGATCAGTGTATCACATCGC
(x_1^1, x_2^0)	TGCACACGCACGACGATCACATCGC
(x_1^0, x_2^1)	CGATCAGTGTCGCTACAGCGAGCAC
(x_1^1, x_2^1)	TGCACACGCACGACGCGCTACAGCGAGCAC

Table 4
DNA sequences chosen to represent all possible choices of subsets

5′-GATCAGCACTGCTAGTCACATAGTGTAGCGAGATATGTCT-3′ 3′-CTAGTCGTGACGATCAGTGTATCACATCGCTCTATACAGA-5′
5′-ATCGACGCATCGCTCACGTGTGCGTGCTGCTAGTGTAGCGAGATATGTCT-3′ 3′-TAGCTGCGTAGCGAGTGCACACGCACGACGATCACATCGCTCTATACAGA-5′
5′-GATCAGCACTGCTAGTCACAGCGATGTCGCTCGTGCAGCGAGACGAGACA-3′ 3′-CTAGTCGTGACGATCAGTGTCGCTACAGCGAGCACGTCGCTCTGCTCTGT-5′
5′-ATCGACGCATCGCTCACGTGTGCGTGCTGCGCGATGTCGCTCGTGCAGCGAGACGAGACA-3′ 3′-TAGCTGCGTAGCGAGTGCACACGCACGACGCGCTACAGCGAGCACGTCGCTCTGCTCTGT-5′

the rule of DNA code design, the DNA sequences satisfy the constraint that there is no runs of more than 4 A's, 4 T's, 4 C's or 4 G's (Roweis et al., 1999; Braich et al., 2003). This indicates that long homopolymer tracts may have unusual secondary structure and the melting temperatures of probe-library hybrids will be more uniform if none of the probe-library hybrids involves long homopolymer tracts. The DNA sequences from the rule of DNA code design also satisfy the constraint that every probe sequence has at least four mismatches with all continuous 15 base alignment of any library sequence (except for with its matching value sequence) (Roweis et al., 1999; Braich et al., 2003). This implies that the constraint ensures that probes bind only weakly where they are not intended to bind. Due to the rule of DNA code design, the DNA sequences satisfy the constraint that every 15 base subsequence of a library sequence has at least four mismatches with all continuous 15 base alignment of itself or any other library sequence (Roweis et al., 1999; Braich et al., 2003). This indicates that the constraint ensures that library strands have a low affinity for themselves. From (Roweis et al., 1999; Braich et al., 2003), errors in the separation of the library strands are errors in the computation. The design is to ensure that the DNA sequences have little secondary structure that might inhibit intended probe-library hybridization. Similarly, the design is also to exclude the DNA sequences that may encourage unintended probe-library hybridization.

Therefore, the hybridization process and the ligation process could be emulated with lower rate of errors. The tube for Algorithm 1 is yielded in simulations. Thus, for S and C a test tube is shown in Table 4.

In simulations of Algorithm 1, a file is applied to store sequences of DNA. Step 1 of Algorithm 1 in simulations is to read the sequences from the file and the sequences are stored in arrays. It is very clear from the definition of set cover that the number of simulation for Step 2 is the number of subsets in a collection. In Step 2a of the first simulation, two distinct arrays are employed to store the result generated by Step 2a. According to definition of set cover, the number of simulation for Step 2b is the number of elements in a subset in a collection. In Steps 2c, 2d and 2e of simulation, the jth element in the ith subset is appended onto every strand and merged in arrays. Repeat simulation of Steps 2c, 2d and 2e of until all of the elements in the ith subset are processed. Therefore, the strands generated by Steps 2c to 2e contain all of the elements in the ith subset and are stored in arrays. Because all of the elements in the ith subset are processed, hence, the choices of subsets containing the elements and the choices of subset not including the elements are both merged in arrays in Step 2f of simulation. The same processing is applied to deal with other subsets. Thus, after every simulation for each subset is finished, all possible choices of subsets consisting of the elements are generated and stored in arrays. So, the result for S and C is shown in Table 5.

It is very obvious from the definition of set cover that the number of simulation for Step 3 of Algorithm 1 is the number of elements. Therefore, two distinct arrays are yielded in Step 3a of simulation. One array including legal choices of subsets and another containing illegal choices of subsets are yielded. The illegal array is deleted in Step 3b of simulation. After every element is checked, the array consisting of every element is produced in Steps 3a and 3b of simulation. Hence, the result is shown in Table 6.

The legal choices of subsets in the array appear in the form of single stranded DNA. Therefore, the aim

Table 5
DNA sequences chosen to represent all possible choices of subsets consisting of the elements

5′-GATCAGCACTGCTAGTCACATAGTGTAGCGAGATATGTCT-3′ 3′-CTAGTCGTGACGATCAGTGTATCACATCGCTCTATACAGA-5′
5′-ATCGACGCATCGCTCACGTGTGCGTGCTGCTAGTGTAGCGAGATATGTCTGTATCGATCAGTGATGTGACATAGAGTAGA-3′ 3′-TAGCTGCGTAGCGAGTGCACACGCACGACGATCACATCGCTCTATACAGACATAGCTAGTCACTACACTGTATCTCATCT-5′
5′-GATCAGCACTGCTAGTCACAGCGATGTCGCTCGTGCAGCGAGACGAGACAACTGTACGAGTGTCTATAGACTCAGTGCAC-3′ 3′-CTAGTCGTGACGATCAGTGTCGCTACAGCGAGCACGTCGCTCTGCTCTGTTGACATGCTCACAGAGATCTGAGTCACGTG-5′
5′-ATCGACGCATCGCTCACGTGTGCGTGCTGCGCGATGTCGCTCGTGCAGCGAGACGAGACAGTATCGATCAGTGATGT-GACATAGAGTAGAACTGTACGAGTGTCTATAGACTCAGTGCAC-3′ 3′-TAGCTGCGTAGCGAGTGCACACGCACGACGCGCTACAGCGAGCACGTCGCTCTGCTCTGTCATAGCTAGTCAC-TACACTGTATCTCATCTTGACATGCTCACAGAGATCTGAGTCACGTG−5′

Table 6
Single stranded DNA sequence chosen to represent the only legal choice

5′-ATCGACGCATCGCTCACGTGTGCGTGCTGCGCGATGTCGCTCGTGCAGCGAGACGAGACAGTATCGATCAGT-GATGTGACATAGAGTAGAACTGTACGAGTGTCTATAGACTCAGTGCAC-3′

Table 7
Double stranded DNA sequence chosen to represent the only legal choice

5′-ATCGACGCATCGCTCACGTGTGCGTGCTGCGCGATGTCGCTCGTGCAGCGAGACGAGACAGTATCGATCAGT-GATGTGACATAGAGTAGAACTGTACGAGTGTCTATAGACTCAGTGCAC-3′ 3′-TAGCTGCGTAGCGAGTGCACACGCACGACGCGCTACAGCGAGCACGTCGCTCTGC-TCTGTCATAGCTAGTCACTACACTGTATCTCATCTTGACATGCTCACAGAGATCTGAGTCACGTG-5′

Table 8
DNA sequence chosen to represent the only minimum-size set cover

5′-ATCGACGCATCGCTCACGTGTGCGTGCTGCGCGATGTCGCTCGTGCAGCGAGACGAGACAGTATCGATCAGTGATGT-GACATAGAGTAGAACTGTACGAGTGTCTATAGACTCAGTGCAC-3′ 3′-TAGCTGCGTAGCGAGTGCACACGCACGACGCGCTACAGCGAGCACGTCGCTCTGCTCTGTCATAGCTAGTCACTACACTGTA-TCTCATCTTGACATGCTCACAGAGATCTGAGTCACGTG-5′

of Step 4 is to form double stranded DNA to legal solutions. So, this process is simulated by explicitly producing the corresponding complementary sequences for the single strands of DNA. Hence, the result is shown in Table 7.

In Step 5, the *shortest* sequence from the array consisting of legal choices of subsets is chosen as a minimum-size set cover. The shortest sequence is a minimum-size set cover because in the proposed encoding scheme the code length of every subset or every element is set to 30 if the corresponding value is equal to one. Therefore, this process is simulated by explicitly finding the shortest sequence from the array consisting of legal choices of subsets. Therefore, the result is shown in Table 8.

The goal of Step 6 is to check whether there are the shortest sequences. Thus, this process is simulated by explicitly examining whether the array is empty. This process generates a resulted value '*yes*', if the array is non-empty. A resulted value, '*no*', is generated by this process if the array is empty. If this process generates a resulted value '*yes*', then in Step 6a of simulation, a legal choice of subsets from the array is selected. Therefore, the minimum-size set cover for S and C is $\{\{1\}, \{2\}\}$. Simulations of Algorithm 2 are similar to those of Algorithm 1.

6. Conclusions

Because the size of solution space in the Adleman–Lipton model is exponential, hence, this limits that we can resolve the size of the NP-complete problem. The main result of this paper shows that the Adleman–

Lipton model can be applied towards developing DNA algorithms to resolving the set cover problem and the problem of exact cover by 3-sets. Furthermore, this work represents clear evidence for the ability of DNA-based computing to solve NP-complete problems.

Famous Cook's Theorem (Cormen et al., 2003; Garey and Johnson, 1979) is that if one algorithm for one NP-complete problem will be developed, then other problems will be solved by means of reduction to that problem. This theorem has been proved to be correct for electronic digit computers. On the other hand, Lipton proposed the DNA algorithm for solving the 3-SAT problem (the first NP-Complete problem) (Lipton, 1995). The three-dimensional matching problem can be reduced to the 3-SAT problem (Garey and Johnson, 1979). But Lipton's algorithm cannot be used to resolving the three-dimensional matching problem. We proposed the DNA algorithm for dealing with the three-dimensional matching problem (Chang and Guo, 2002c), which is different from Lipton's 3-SAT DNA algorithm. Another famous NP-complete problem is the set-partition problem. The set-partition problem can be reduced to the three-dimensional matching problem (Garey and Johnson, 1979). But our algorithm cannot be applied to solving the set-partition problem. The set-partition problem is still not solved. So, we are not sure whether Cook's Theorem is also correct for *molecular* computers and we are also not sure whether molecular computing can be applied to dealing with every NP-complete problem.

It is indicated from (Garey and Johnson, 1979) that there are lots of NP-complete problems containing mathematical operations. It is very difficult from (Adleman, 1994) that using molecular computing finishes mathematical operations. This is to say that solving those NP-complete problems with mathematical operations is open questions and is also our future researching directions.

References

Adleman, L., 1994. Molecular computation of solutions to combinatorial problems. Science 266, 1021–1024.

Adleman, L.M., 1996. On constructing a molecular computer. DNA based computers. In: Lipton, R., Baum, E. (Eds.), DIMACS: Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society, pp. 1–21.

Amos, M., 1997. DNA Computation, Ph.D. Thesis, Department of Computer Science, The University of Warwick.

Arita, M., Suyama, A., Hagiya, M., 1997. A heuristic approach for Hamiltonian path problem with molecules. In: Proceedings of Second Genetic Programming (GP-97). pp. 457–462.

Bach, E., Condon, A., Glaser, E., Tanguay, C., 1996. DNA models and algorithms for NP-complete problems. In: Proceedings of the Eleventh Annual Conference on Structure in Complexity Theory. pp. 290–299.

Boneh, D., Dunworth, C., Lipton, R.J., Sgall, J., 1996. On the computational power of DNA. In: Discrete Applied Mathematics. Special Issue on Computational Molecular Biology, vol. 71. pp. 79–94.

Braich, R.S., Johnson, C., Rothemund, P.W.K., Hwang, D., Chelyapov, N., Adleman, L.M., Solution of a satisfiability problem on a gel-based DNA computer. In: Proceedings of the Sixth International Conference on DNA Computation in the Springer-Verlag Lecture Notes in Computer Science series.

Chang, W.-L., Guo, M., 2002. Solving the Dominating-set Problem in Adleman–Lipton's Model. In: Proceedings of the Third International Conference on Parallel and Distributed Computing, Applications and Technologies, Japan.

Chang, W.-L., Guo, M., 2002. Solving the Clique Problem and the Vertex Cover Problem in Adleman–Lipton's Model. In: Proceedings of the IASTED International Conference, Networks, Parallel and Distributed Processing, and Applications, Japan.

Chang, W.-L., Guo, M., 2002. Resolving the 3-dimensional matching problem and the set packing problem in Adleman–Lipton's Model. In: Proceedings of the IASTED International Conference, Networks, Parallel and Distributed Processing, and Applications, Japan.

Chang, W.-L., Chu, C.-P., Huang, S.-C., 2002. DNA solution of the independent-set problem. In: Proceedings of the Eighth Workshop on High Performance Computing, Taiwan, pp. 120–126.

Chang, W.-L., Guo, M., 2003. Using Sticker for Solving the Dominating-set Problem in the Adleman–Lipton Model. IEICE Transaction on Information and System. In press.

Cormen, T.H., Leiserson, C.E., Rivest, R.L. Introduction to Algorithms.

Fu, B., 1997. Volume Bounded Molecular Computation, Ph.D. Thesis, Department of Computer Science, Yale University.

Garzon, M.H., Deaton, R.J., 1999. Biomolecular computing and programming. IEEE Trans. Evolution. Comput. 3, 236–250.

Garey, M.R., Johnson, D.S., 1979. Computer and Intractability. Freeman, San Fransico, CA.

LaBean, T.H., Winfree, E., Reif, J.H., 2000. Experimental progress in computation by self-assembly of DNA tilings. Theoret. Computer Sci. 54, 123–140.

LaBean, M.C., Reif, T.H., Seeman, J.H. Logical computation using algorithmic self-assembly of DNA triple-crossover molecules. Nature 407, 493–496.

Lipton, R.J., 1995. DNA solution of hard computational problems. Science 268, 542–545.

Morimoto, N., Arita, M., Suyama, A., 1999. Solid phase DNA solution to the Hamiltonian path problem. DIMACS (Series in Discrete Mathematics and Theoretical Computer Science), vol. 48. pp. 93–206.

Narayanan, A., Zorbala, S., 1998. DNA algorithms for computing shortest paths. In: Koza, J.R. et al. (Eds.), Genetic Programming. Proceedings of the Third Annual Conference. pp. 718–724.

Paun, G., Rozenberg, G., Salomaa, A., 1998. DNA Computing: New Computing Paradigms. Springer, New York. ISBN: 3-540-64196-3.

Perez-Jimenez, M.J., Sancho-Caparrini, F., 2001, Solving Knapsack Problems in a Sticker Based Model. In: Proceedings of the Seventh Annual Workshop on DNA Computing, DIMACS: Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society.

Quyang, Q., Kaplan, P.D., Liu, S., Libchaber, A., 1997. DNA solution of the maximal clique problem. Science 278, 446–449.

Reif, J.H., LaBean, T.H., Seeman, 2000. Challenges and Applications for Self-Assembled DNA-Nanostructures. In: Proceedings of the Sixth DIMACS Workshop on DNA Based Computers, Leiden, Holland, June 2000.

Roweis, S., Winfree, E., Burgoyne, R., Chelyapov, N.V., Goodman, M.F., Rothemund, P.W.K., Lipton, L.M., 1999. A Sticker Based Model for DNA Computation. In: Landweber, L., Baum, E. (Eds.), DIMACS: Series in Discrete Mathematics and Theoretical Computer Science. Proceedings of the Second Annual Workshop on DNA Computing, Princeton University. American Mathematical Society, pp. 1–29.

Shin, S.-Y., Zhang, B.-T., Jun. S.-S., 1999. Solving traveling salesman problems using molecular programming. In: Proceedings of the 1999 Congress on Evolutionary Computation (CEC99), vol. 2. pp. 994–1000.

Sinden, R.R., 1994. DNA Structure and Function. Academic Press, New York.