Available online at www.sciencedirect.com
SCIENCE DIRECT®
Future Generation Computer Systems 20 (2004) 875–885
FGCS FUTURE GENERATION COMPUTER SYSTEMS
www.elsevier.com/locate/future

Towards solution of the set-splitting problem on gel-based DNA computing

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

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

Received 9 May 2003; received in revised form 9 September 2003; accepted 22 October 2003

Available online 19 March 2004

Abstract

Adleman wrote the first paper that demonstrated that DNA (*DeoxyriboNucleic Acid*) strands could be applied for dealing with solutions of the NP-complete Hamiltonian path problem (HPP). Lipton wrote the second paper that showed that the Adleman techniques could also be used to solve the NP-complete satisfiability (SAT) problem (the first NP-complete problem). Adleman and his co-authors proposed *sticker* for enhancing the Adleman–Lipton model. In this paper, it proves how to apply sticker in the sticker-based model to construct solution space of DNA in the *set-splitting problem* and how to apply DNA operations in the Adleman–Lipton model to solve that problem from the solution space of sticker.

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

1. Introduction

Through advances in molecular biology [1], it is now possible to produce roughly 10^{18} DNA strands that fit in a test tube. Adleman [2] wrote the first paper that showed how DNA strands could be applied to manipulate solutions for an instance of the NP-complete Hamiltonian path problem (HPP). Lipton [3] wrote a second paper that demonstrated that the Adleman techniques could be employed to solve the NP-complete satisfiability problem (the first NP-complete problem). Adleman and co-workers [14] proposed *sticker* for enhancing the error rate of hybridization in the Adleman–Lipton model.

In this paper, we use *sticker* in the sticker-based model to construct solution spaces of DNA strands for the *set-splitting problem*. Then by applying biological operations to the Adleman–Lipton model, we develop a DNA-based algorithm. We also show that using our proposed DNA-based algorithm for the solution spaces of DNA strands solves the set-splitting problem. Furthermore, this work presents clear evidence of the ability of molecular computing to solve the NP-complete problem.

This paper is organized as follows. In Section 2, the Adleman–Lipton model is introduced and its comparison is made with other models. Section 3 introduces a DNA algorithm for solving the set-splitting problem from solution spaces of sticker in the Adleman–Lipton

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

0167-739X/$ – see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2003.10.010

model. In Section 4, the experimental results of simulated DNA computing are given. Conclusions are drawn in Section 5.

2. DNA model of computation

2.1. The Adleman–Lipton model

A DNA (*DeoxyriboNucleic Acid*) strand is a polymer, which is strung together from monomers called *DeoxyriboNucleotides* [1,16]. The structure of a nucleotide, cited from [16], is illustrated (in a very simplified way) in Fig. 1, where B is one of the four possible bases (abbreviated as *A*, *G*, *C*, or *T*), P is the phosphate group, and the rest of the "stick" is the sugar base (with its carbons enumerated 1′ to 5′).

The first way to link nucleotides together is for the 5′-phosphate group of one nucleotide to join with the 3′-hydroxyl group of the other, forming a *phosphodiester* bond. The resulting molecule has the 5′-phosphate group of one nucleotide, denoted as 5′-end, and the 3′-OH group of the other nucleotide, denoted as 3′-end, available for bonding. This gives the molecule *directionality*, and we can talk about the direction of the 5′-end to the 3′-end or the 3′-end to the 5′-end. The second way to link them together is for the base of one nucleotide to interact with the base of the other nucleotide to form a *hydrogen* bond. This bonding is subject to the following restrictions on the base pairing: *A* and *T* can pair together, and *C* and *G* can pair together—no other pairings are possible. This pairing principle is called the Watson–Crick complementary.

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, then we say that it is a 20 mer (it is a polymer containing 20 monomers). 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 20 bp. Hybridization is a special technology term for the pairing of two single DNA strands to make a double helix and also takes advantages of the specificity of DNA base pairing for the detection of specific DNA strands [1,16]. (For more discussion of the relevant biological background refers to [1,11,16].)

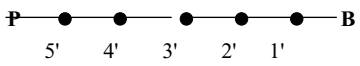

Fig. 1. A schematic representation of a nucleotide.

In the Adleman–Lipton model [2,3], *splints* were used to construct the corresponding edges or paths of a particular graph that represented all possible binary numbers. As it stands, their construction indiscriminately builds all splints that lead to a complete graph. This means that hybridization has a higher probability of errors. Hence, Adleman and co-workers [14] proposed the sticker-based model, which was an abstract of molecular computing based on DNA with a random access memory as well as a new form of encoding the information. (For more discussion of the sticker-based model refer to [14].)

DNA operations for the Adleman–Lipton model [2,3,11,12] are described below. These operations will be used for figuring solutions of the set-splitting problem.

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 sub-strand 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 to the individual strands.
3. *Detect*. Given a tube P, if P includes at least one DNA molecule we say 'yes', and if it contains none we say 'no'.
4. *Discard*. Given a tube P, the operation will discard the tube P.

5. *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. The comparison of the Adleman–Lipton model with other models

Techniques in the Adleman–Lipton model could be applied for solving the NP-complete Hamiltonian path problem and satisfiability (SAT) problem by linearly increasing time and exponentially increasing the volumes of DNA [2,3]. Quyang et al. [4] proved that restriction enzymes could be used to solve the NP-complete clique problem. The maximum number of vertices they can process is limited to 27 because the size of the pool along with the size of the problem exponentially increases [4]. Arita et al. [5] described new molecular experimental techniques for searching a Hamiltonian path. Morimoto et al. [6] offered a solid-phase method to find a Hamiltonian path. Narayanan and Zorbala [7] demonstrated that the Adleman–Lipton model was extended for solving the traveling salesman problem. Shin et al. [8] presented an encoding scheme that applies fixed-length codes for representing integer and real values. Their method could also be employed towards solving the traveling salesman problem. Amos [13] proposed the parallel filtering model for resolving the Hamiltonian path problem, sub-graph isomorphism problem, 3-vertex-colourability problem, clique problem and independent-set problem. Roweis et al. [14] proposed sticker-based model to enhance the Adleman–Lipton model. Their model could be used to determine solutions for an instance of the set cover problem. Perez-Jimenez and Sancho-Caparrini [15] employed sticker-based model [14] to resolve the knapsack problem. Fu [21] proposed new algorithms to resolve 3-SAT, 3-Coloring and the independent set. In our previous work, Chang and Guo [17–20] proved how the DNA operations from solution space of *splint* in the Adleman–Lipton model could be employed to develop DNA algorithms. Those DNA algorithms could be applied for solving the dominating-set problem, vertex cover problem, clique problem, independent-set problem, 3-dimensional matching problem and set-packing problem. In our previous work, Chang and co-workers [25,26] employed the sticker-based model and the Adleman–Lipton model to deal with the dominating-set problem and the set-basis problem for decreasing the error rate of hybridization.

3. Using sticker for solving the set-splitting problem in the Adleman–Lipton model

3.1. Definition of the set-splitting problem

Assume that a finite set S is $\{s_1, \ldots, s_d\}$, where s_e is the e^{th} element for $1 \leq e \leq d$ in S. $|S|$ is denoted as the number of elements in S and $|S|$ is equal to d. Suppose that a collection C is the set of subsets to a finite set S and is $\{C_1, \ldots, C_f\}$, where C_g is the g^{th} element for $1 \leq g \leq f$ in C. $|C|$ is denoted as the number of subsets in C and $|C|$ is equal to f. Mathematically, the set-splitting problem is to find whether there is a partition of S into two subsets S_1 and S_2 such that no subset in C is entirely contained in either S_1 or S_2 [10]. The problem has been proved to be NP-complete problem [10].

There are a finite set S and a collection C of subsets for S in Fig. 2. The finite set S is $\{1, 2\}$ and the collection C is $\{\{1, 2\}\}$. The two sets define such a problem. The set splitting for S and C in Fig. 2 is $S_1 = \{1\}$ and $S_2 = \{2\}$ or $S_1 = \{2\}$ and $S_2 = \{1\}$. It is indicated from [10] that finding a set splitting is a NP-complete problem, so it can be formulated as a "search" problem.

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

3.2. Using sticker for constructing solution space of DNA sequence for the set-splitting problem

In the Adleman–Lipton model, their main idea is to first generate solution space of DNA sequences for those problems solved. Then, basic biological operations are used to select legal solution and to remove illegal solution from solution space. Therefore, a finite

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

Fig. 2. A finite set S and a collection C of subsets for S.

Table 1
The solution space of the subsets for the finite S in Fig. 2

2-Digit binary number	Corresponding subset
00	$\emptyset$
01	$\{1\}$
10	$\{2\}$
11	$\{2, 1\}$

set S with d elements and a collection C with f elements for the subsets of the finite set S, the first step of solving the set-splitting problem is to produce a test tube, which includes all possible subsets from the finite set. Assume that a d-digit binary number represents each possible subset for S. Also suppose that S_1 is a subset of S. If the i^{th} bit in a d-digit binary number is set to 1, then it represents that the i^{th} element in S is in S_1. If the i^{th} bit in a d-digit binary number is set to 0, then it represents the corresponding element is not in S_1. By doing it this way, all possible subsets of S are transformed into an ensemble of all d-digit binary numbers.

Hence, Table 1 denotes the solution space of the subsets for the finite set S in Fig. 2. The binary number, 00, in Table 1 represents the corresponding subset to be empty. The binary numbers, 01 and 10, in Table 1 represent those corresponding subsets $\{1\}$ and $\{2\}$. The binary number, 11, in Table 1 represents the corresponding subset to be $\{2, 1\}$. Though there are four 2-digit binary numbers for representing four possible subsets in Table 1, not every 2-digit binary number corresponds to a *legal* solution. In the next subsection, basic biological operations are used to develop an algorithm for removing illegal subsets and determining legal solutions.

Collection C with f elements is a collection of subsets from S. Therefore, every subset in C is represented by the same method as above. Table 2 denotes representation of each subset in the collection C of Fig. 2. The only subset, $\{1, 2\}$, in Table 2 is represented as the 2-digit binary number, 11.

Table 2
Denote representation of each subset in the collection C in Fig. 2

Subset	Corresponding 2-digit binary representation
$\{1, 2\}$	11

To implement this, assume that an unsigned integer X is represented by a binary number $x_d, x_{d\text{-}1}, \ldots, x_1$, where the value of x_i is 1 or 0 for $1 \leq i \leq d$. The integer X contains 2^d kinds of possible values. Each possible value represents a subset for a finite set S. Therefore, it is clear that an unsigned integer X forms 2^d possible subsets. A bit x_i in an unsigned integer X represents the ith element in S. If the i^{th} element is in a subset, then the value of x_i is set to 1. If the i^{th} element is out of a subset, then the value of x_i is set to 0.

To represent all possible subsets for a finite set S with d elements for the set-splitting problem, *sticker* [14,22] is used to construct solution space for that problem to be solved. For every bit x_i and $1 \leq i \leq d$, two *distinct* 15 base value sequences were designed. The value of x_i can be 0 or 1. 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 0. Each of the 2^d possible subsets is represented by a library sequence of $15 \times d$ bases consisting of the concatenation of one value sequence for each bit. DNA molecules with library sequences are termed library strands and a combinatorial pool containing library strands is termed a library. The probes used for separating the library strands have sequences complementary to the value sequences. Because a collection C is the set of subsets from a finite set S, every element in C is a subset from S. Therefore, the same DNA sequences above are also applied to represent every element in C.

It is pointed out from [14,22] that errors in the separation of the library strands are errors in the computation. Sequences must be designed to ensure that library strands have little secondary structure that might inhibit intended probe-library hybridization. The design must also exclude sequences that might encourage unintended probe-library hybridization. To help achieve these goals, sequences were computer-generated to satisfy the following constraint [22].

1. Library sequences contain only A's, T's, and C's.
2. All library and probe sequences have no occurrence of five or more consecutive identical nucleotides, i.e. no runs of more than 4 A's, 4 T's, 4 C's or 4 G's occur in any library or probe sequences.
3. Every probe sequence has at least 4 mismatches with all 15 base alignment of any library sequence (except with its matching value sequence).

4. Every 15 base subsequence of a library sequence has at least 4 mismatches with all 15 base alignment of itself or any other library sequence.
5. No probe sequence has a run of more than 7 matches with any 8 base alignment of any library sequence (except for with its matching value sequence).
6. No library sequence has a run of more than 7 matches with any 8 base alignment of itself or any other library sequence.
7. Every probe sequence has 4, 5, or 6 G's in its sequence.

Constraint (1) is motivated by the assumption that library strands composed only of A's, T's, and C's will have less secondary structure than those composed of A's, T's, C's, and G's [23]. Constraint (2) is motivated by two assumptions: first, that long homopolymer tracts may have unusual secondary structure and second, that the melting temperatures of probe-library hybrids will be more uniform if none of the probe-library hybrids involve long homopolymer tracts. Constraints (3) and (5) are intended to ensure that probes bind only weakly where they are not intended to bind. Constraints (4) and (6) are intended to ensure that library strands have a low affinity for themselves. Constraint (7) is intended to ensure that intended probe-library pairings have uniform melting temperatures.

The Adleman program [22] was modified for generating those DNA sequences to satisfy the constraints above. For example, the two elements in the finite set S of Fig. 2, the DNA sequences generated were: $x_1^0 = AAAACTCACCCTCCT$, $x_2^0 = TCTAATATAATTACT$, $x_1^1 = TTTCAATAACACCTC$ and $x_2^1 = ATTCACTTCTTTAAT$. Because the only subset in the collection C of Fig. 2 includes the first element and the second element in S, two 15 base DNA sequences, $ATTCACTTCTTTAAT$ (x_2^1) and $TTTCAATAACACCTC$ (x_1^1) are used for representing them. For every possible subset from the finite set S of Fig. 2, the corresponding library strand was synthesized by employing a mix-and-split combinatorial synthesis technique [24]. Similarly, for any d-element set, all of the library strands to represent every possible subset could also be synthesized using the same technique.

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

The following pseudo-algorithm explains how to solve the *set-splitting problem*:

(1) Generate solution space of DNA sequences to encode 2^d subsets for any d-element set, S.
(2) Keep only those DNA sequences that represent subsets that do not entirely contain any subset in a collection C.
(3) If any DNA sequences remain, then we have a "yes" to read an answer. Otherwise we have a "no".

The finite set S and the collection C in Fig. 2 are applied to explain the processing of the pseudo-algorithm for solving the set-splitting problem. From Step 1 in the pseudo-algorithm, four DNA sequences are generated for S and C. They encode $\emptyset$, $\{1\}$, $\{2\}$ and $\{2, 1\}$, respectively. The only subset in C is $\{1, 2\}$. Hence, from Step 2 in the pseudo-algorithm, legal DNA sequences are kept. The legal DNA sequences represent $\{1\}$ and $\{2\}$, respectively. From Step 3 in the pseudo-algorithm, because legal DNA sequences remain, the answer for the set-splitting problem is found to be $S_1 = \{1\}$ and $S_2 = \{2\}$, or $S_1 = \{2\}$ and $S_2 = \{1\}$ from the finite set S and the collection C in Fig. 2.

The following DNA algorithm is proposed to solve the *set-splitting problem*:

Algorithm 1. Solving the set-splitting problem.

(1) Input (T_0), where tube T_0 includes solution space of DNA sequences to encode all of the possible subsets for any d-element set, S, with those techniques mentioned in Section 3.2.
(2) For $j = 1$ to $|C|$, where $|C|$ is the number of subsets in a collection C.
  (a) For $k = 1$ to $|C_j|$, where $|C_j|$ is the number of elements in C_j that is an element in C.
    Assume that the kth element in C_j is the ith element in S and x_i is used to represent it.
    (b) $T_0 = +(T_0, x_i^1)$ and $T_{\mathrm{OFF}} = -(T_0, x_i^1)$.
    (c) $T_{\mathrm{ON}} = \cup(T_{\mathrm{OFF}}, T_{\mathrm{ON}})$.
  EndFor
  (d) Discard(T_0).
  (e) $T_0 = \cup(T_{\mathrm{ON}}, T_0)$.
EndFor

(3) For $j = 1$ to $|C|$, where $|C|$ is the number of subsets in a collection C.
(a) For $k = 1$ to $|C_j|$, where $|C_j|$ is the number of elements in C_j that is an element in C.
Assume that the kth element in C_j is the ith element in S and x_i is used to represent it.
(b) $T_0 = +(T_0, x_i^0)$ and $T_{\mathrm{OFF}} = -(T_0, x_i^0)$.
(c) $T_{\mathrm{ON}} = \cup(T_{\mathrm{OFF}}, T_{\mathrm{ON}})$.
EndFor
(d) Discard(T_0).
(e) $T_0 = \cup(T_{\mathrm{ON}}, T_0)$.
EndFor
(4) If Detect(T_0) == "yes" then
(a) Read (T_0).

It is obvious from the steps in Algorithm 1 that the set-splitting problem for any d-element set can be solved.

Proof. In Step 1, a test tube of DNA strands, that encode all 2^d possible input bit sequences $x_d, \ldots, x_1$, is generated. It is clear that the test tube includes all 2^d possible subsets for any d-element set, S.

Step 2 contains one outer loop and one inner loop. The outer loop will execute $|C|$ times, where $|C|$ is the number of subsets in a collection C. The inner loop will execute $(|C_j| \times |C|)$ times, where $|C_j|$ is the number of elements in C_j that is an element in C. According to the definition of set-splitting [9,10], it is to find whether there is a partition of S within two subsets S_1 and S_2 such that no subset in C is entirely contained in either S_1 or S_2. Thus, the first execution of Step 2b applies "extraction" operation to form two test tubes: T_0 and T_{OFF}. The first tube T_0 contains all of the strands that have $x_i = 1$. The second tube T_{OFF} consists of all of the strands that have $x_i = 0$. Tube T_0 represents those partitions, which contains the element s_i. Tube T_{OFF} represents those partitions, which do not include the element s_i. That means element s_i is in S_1 but not in S_2. Then, the first execution of Step 2c uses the "merge" operation to pour two tubes, T_{OFF} and T_{ON}, into tube T_{ON}. That means tube T_{ON} obtains the strands from tube T_{OFF}. After Steps 2b and 2c are repeated to execute $(|C_j|)$ times, tube T_0 includes the partitions, which contain every element in C_j that is an element in C. Tube T_{ON} consists of the partitions which do not include every element in C_j. It is indicated from definition of set splitting that the strands in T_0 represent illegal partitions. Hence, Step 2d uses the "discard" operation to discard the tube T_0. Since the strands in tube T_{ON} possible represent legal partitions, Step 2e applies the "merge" operation to pour two tubes, T_0 and T_{ON} into tube, T_0. That means that tube T_0 obtains the strands in the tube T_{ON}. For other subsets in C, similar processing is also finished. Therefore, after all of the second steps are processed, tube T_0 includes the partitions, which do not entirely contain every element in every subset in C.

Step 3 includes one outer loop and one inner loop. The outer loop will execute $|C|$ times, where $|C|$ is the number of subsets in a collection C. The inner loop will execute $(|C_j| \times |C|)$ times, where $|C_j|$ is the number of elements in C_j that is an element in C. Due to definition of set-splitting [9,10], the first execution of Step 3b applies the "extraction" operation to form two test tubes: T_0 and T_{OFF}. The first tube T_0 contains all of the strands that have $x_i = 0$. The second tube T_{OFF} consists of all of the strands that have $x_i = 1$. Tube T_0 represents those partitions, which do not include element s_i. Tube T_{OFF} represents those partitions, which contain element s_i. This means that element s_i is in S_2 and out of S_1. Hence, the first execution of Step 3c uses the "merge" operation to pour two tubes, T_{OFF} and T_{ON} into tube, T_{ON}. That means that tube T_{ON} obtains the strands from tube T_{OFF}. After Steps 3b and 3c are repeated to execute $(|C_j|)$ times, tube T_0 includes the partitions, which contains every element in C_j that is an element in C. Tube T_{ON} consists of the partitions, which do not include every element in C_j. It is indicated from definition of set splitting that strands in T_0 represent illegal partitions. Hence, Step 3d uses the "discard" operation to discard tube T_0. Since strands in tube T_{ON} possible represent legal partitions, Step 3e applies the "merge" operation to pour two tubes, T_0 and T_{ON} into tube, T_0. That means tube T_0 obtains the strands from tube T_{ON}. For other subsets in C, similar processing is also finished. Therefore, after all of the third steps are processed, tube T_0 includes the partitions, which do not include every element in every subset in C.

Step 4 uses the "detect" operation to detect tube T_0. If there is any strand in tube T_0, then Step 4a employs the "read" operation to describe the 'sequence' of a molecular in the tube T_0. Hence, the answer for the set-splitting problem is found and described.

The finite set S and the collection C of Fig. 2 are used to show the power of Algorithm 1. It is pointed out in Step 1 of Algorithm 1 that tube T_0 is filled with four library strands with those techniques mentioned in Section 3.2, representing four possible subsets from set S in Fig. 2. The number of subsets in C of Fig. 2 is one, so the number of executions of the outer loop in Step 2 of Algorithm 1 is one time. The number of elements in this one subset, which is in collection C, is two. Therefore, the number of executions of the inner loop in Step 2 of Algorithm 1 is two times.

According to the first execution of Step 2b of Algorithm 1, two tubes are generated. The first tube, T_0, contains subsets $\{1\}$ and $\{1, 2\}$ and the second tube, T_{OFF}, also contains subsets $\emptyset$ and $\{2\}$. The first execution of Step 2c in Algorithm 1 pours two tubes T_{OFF} and T_{ON} into tube T_{ON}. Therefore, tube T_{ON} now contains additional subsets $\emptyset$ and $\{2\}$. It is clear from the second execution of Step 2b that two tubes are yielded. The first tube, T_0 contains subset $\{1, 2\}$. The second tube, T_{OFF}, contains subset $\{1\}$. It is indicated from the second execution of Step 2c that tube T_{ON} contains subsets $\emptyset$, $\{1\}$ and $\{2\}$. In light of the definition for set splitting, tube T_0 contains illegal partition, the first execution of Step 2d applies the "discard" operation to discard tube T_0. After the first execution of Step 2e the "merge" operation, tube T_0 contains subset $\emptyset$, $\{1\}$ and $\{2\}$.

Since the number of subsets in C of Fig. 2 is one, the number of executions of the outer loop in Step 3 of Algorithm 1 is one time. The number of elements in this one subset, which is in collection C, is two. Therefore, the number of executions for the inner loop of Step 3 is two times. During the first execution of Step 3b, two tubes are generated. The first tube, T_0, contains subsets $\emptyset$ and $\{2\}$ and the second tube, T_{OFF}, contains subset $\{1\}$. Next, the first execution of Step 3c pours two tubes T_{OFF} and T_{ON} into tube T_{ON}. Therefore, tube T_{ON} now contains subset $\{1\}$. It is clear from the second execution of Step 3b that two tubes are yielded. The first tube T_0 contains subset $\emptyset$. The second tube T_{OFF} contains subset: $\{2\}$. It is indicated from the second execution of Step 3c that tube T_{ON} contains subsets $\{1\}$ and $\{2\}$. Based upon the definition of set splitting, tube T_0 contains the illegal partition. Therefore, the first execution of Step 3d applies the "discard" operation to discard tube T_0. After the first execution of Step 3e the "merge" operation, tube T_0 contains subset $\{1\}$ and $\{2\}$.

The first execution of Step 4 applies the "detect" operation to detect tube T_0. Because tube T_0 is not empty, Step 4a employs the "read" operation to describe the 'sequence' of a molecular in tube T_0. The answer for the set-splitting problem is found to be $S_1 = \{1\}$ and $S_2 = \{2\}$ or $S_1 = \{2\}$ and $S_2 = \{1\}$ from the finite set S and the collection C in Fig. 2. □

3.4. The complexity of the proposed DNA algorithm

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

Theorem 1. *The set-splitting problem for any d-element set S and any f-subset collection C can be solved with* $O(d \times f)$ *biological operations in the Adleman–Lipton model.*

Proof. Algorithm 1 can be applied for solving the set-splitting problem for any d-element set S and any f-subset collection C. Algorithm 1 includes three main steps. It is indicated from Step 2 in Algorithm 1 that it takes $(d \times f)$ "extraction" operations, $(d \times f + f)$ "merge" operations and f "discard" operations. From Step 3 of Algorithm 1, it takes $(d \times f)$ "extraction" operations, $(d \times f + f)$ "merge" operations and f "discard" operations. From Step 4, it takes one "detect" operation and one "read" operation. Hence, it is inferred that the time complexity of Algorithm 1 is $O(d \times f)$ biological operations in the Adleman–Lipton model. □

Theorem 2. *The set-splitting problem for any d-element set S and any f-subset collection C can be solved with sticker to construct* $O(2^d)$ *library strands in the Adleman–Lipton model.*

Proof. Refer to Theorem 1. □

Theorem 3. *The set-splitting problem for any d-element set S and any f-subset collection can be solved with three tubes in the Adleman–Lipton model.*

Proof. Refer to Theorem 1. □

Theorem 4. *The set-splitting problem for any d-element set S and any f-subset collection can be solved with the longest library strand,* $O(15 \times d)$*, in the Adleman-Lipton model.*

Proof. Refer to Theorem 1. □

4. Experimental results of simulated DNA computing

We modified the Adleman program [22]. This modified program was applied to generate DNA sequences for solving the set-splitting problem for any *d*-element set *S* and any *f*-subset collection *C*. We also added subroutines to the Adleman program for simulating biological operations in the Adleman–Lipton model in Section 2. We added subroutines to the Adleman program to simulate Algorithm 1 in Section 3.3. For any *d*-element set *S* and any *f*-subset collection *C*, the size of the library strands is 2^d. Due to the limit of memory space and hard-disk space, the value of *d* was less than or equal to 20. The program shown in Algorithm 1 has been abbreviated for this article. The full program is available upon request from the authors.

The Adleman program is used to construct each 15-base DNA sequence for each bit of the library. For each bit, the program is applied to generate two 15-base random sequences ('1' or '0') and checking to see if the library strands satisfy the seven constraints in Section 3.2 with the new DNA sequences added. If the constraints are satisfied, the new DNA sequences are 'greedily' accepted. If the constraints are not satisfied then mutations are introduced one by one into the new block until either 'the constraints are satisfied and then the new DNA sequences are accepted' or 'a threshold for the number of mutations is exceeded', the program fails and it exits'. If *d*-bits that satisfy the constraints are found then the program has succeeded and it outputs these sequences.

Consider the finite set *S* and the collection *C* in Fig. 2. The finite set *S* contains $\{1, 2\}$ and the collection *C* contains $\{\{1, 2\}\}$. DNA sequences generated by the modified Adleman program are shown in Table 3. The program takes two mutations to make new DNA sequences for the two elements in *S*. With the nearest neighbor parameters, the program was used to calculate the enthalpy, entropy, and free energy for binding each probe to its corresponding region on a library strand. The energy levels are shown in Table 4. Only G really matters to the energy of each bit. For example, the delta G for the probe binding a '1' in the first bit is estimated to be 24.3 kcal/mol and the delta G for the probe binding a '0' is estimated to be 27.5 kcal/mol.

Table 3
Sequences chosen to represent the two elements in *S* in Fig. 2

Vertex	5′ → 3′ DNA sequence
x_2^0	*TCTAATATAATTACT*
x_1^0	*AAAACTCACCCTCCT*
x_2^1	*ATTCACTTCTTTAAT*
x_1^1	*TTTCAATAACACCTC*

Table 4
The energy for binding each probe to its corresponding region in a library strand

Vertex	Enthalpy energy (*H*)	Entropy energy (*S*)	Free energy (*G*)
x_2^0	104.8	283.7	19.9
x_1^0	113.7	288.7	27.5
x_2^1	107.8	283.5	23
x_1^1	105.6	271.6	24.3

The program simulated a mix-and-split combinatorial synthesis technique [24] to synthesize the library strand to every possible subset. The library strands shown in Table 5 represent four possible subsets ∅, $\{1\}$, $\{2\}$ and $\{1, 2\}$. The program also calculates the average and standard deviation for enthalpy, entropy and free energy for all probe and library strand interactions as shown in Table 6. The standard deviation for delta G is small because this is partially enforced

Table 5
DNA sequences chosen represents all possible subsets

5′-*TCTAATATAATTACTAAAACTCACCCTCCT*-3′
3′-*AGATTATATTAATGATTTTGAGTGGGAGGA*-5′
5′-*TCTAATATAATTACTTTTCAATAACACCTC*-3′
3′-*AGATTATATTAATGAAAAGTTATTGTGGAG*-5′
5′-*ATTCACTTCTTTAATAAAACTCACCCTCCT*-3′
3′-*TAAGTGAAGAAATTATTTTGAGTGGGAGGA*-5′
5′-*ATTCACTTCTTTAATTTTCAATAACACCTC*-3′
3′-*TAAGTGAAGAAATTAAAAGTTATTGTGGAG*-5′

Table 6
The energy over all probe/library strand interactions

	Enthalpy energy (H)	Entropy energy (S)	Free energy (G)
Average	107.975	281.875	23.675
Standard deviation	3.48298	6.28739	2.72615

Table 7
DNA sequences generated by Step 2 represent possible partitions

5′-*TCTAATATAATTACTAAAACTCACCCTCCT*-3′
5′-*TCTAATATAATTACTTTTCAATAACACCTC*-3′
5′-*ATTCACTTCTTTAATAAAACTCACCCTCCT*-3′

Table 8
DNA sequences generated by Step 3 represent legal partitions and the answer for the set-splitting problem

5′-*TCTAATATAATTACTTTTCAATAACACCTC*-3′
5′-*ATTCACTTCTTTAATAAAACTCACCCTCCT*-3′

by the constraint that is 4, 5, or 6 G's (the seventh constraint in Section 3.2 of probe sequences).

The Adleman program is employed to compute distribution of the types of potential mishybridizations. Distribution of the types of potential mishybridizations is the absolute frequency of a probe-strand match from bit length k from 0 to bit length 15 (for DNA sequences) where probes are not supposed to match the strands. Distribution is 57, 104, 110, 105, 100, 103, 59, 36, 11, 2, 1, 0, 0, 0, 0 and 0. The last five zeros means there are 0 occurrences where a probe matches a strand at 11, 12, 13, 14 and 15 places. This shows that the third constraint in Section 3.2 has been satisfied. It is clear that the number of matches peaks at 2(1 1 0). This means there are 110 occurrences where a probe matches a strand at two places.

The simulation results for Steps 2 and 3 are shown in Tables 7 and 8. The set-splitting simulation for Step 4 is shown in Table 8. This means that the answer for the set-splitting problem is $S_1 = \{1\}$ and $S_2 = \{2\}$ or $S_1 = \{2\}$ and $S_2 = \{1\}$ from the finite set S and the collection C in Fig. 2.

5. Conclusions

Algorithm 1 for solving the set-splitting problem is based on biological operations using the Adleman–Lipton model and solution space of stickers using the sticker-based model. This algorithm has several advantages over the Adleman–Lipton model and sticker-based model. Firstly, the algorithm actually has a lower rate of errors for hybridization when using the modified Adleman program to generate good DNA sequences for constructing the solution space of stickers for the set-splitting problem. The basic biological operations used in the Adleman–Lipton model were also employed to finish the function of judging a legal partition for solving the set-splitting problem. Secondly, the basic biological operations used in the Adleman–Lipton model have been performed in a fully automated manner in their laboratory. 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-splitting problem, the number of tubes, the longest length of DNA library strands, the number of DNA library strands and the number of biological operations are $O(c)$, $O(15 \times d)$, $O(2^d)$ and $O(d \times f)$, respectively. From the above conclusions, it is obvious that this algorithm can be easily performed in a fully automated laboratory. Furthermore, the same algorithm generates 2^d library strands which satisfy the constraints in [22] and correspond to 2^d possible solutions. This allows the algorithm to be applied to a larger instance of the set-splitting problem.

References

[1] R.R. Sinden, DNA Structure and Function, Academic Press, New York, 1994.
[2] L. Adleman, Molecular computation of solutions to combinatorial problems, Science 266 (1994) 1021–1024.
[3] R.J. Lipton, DNA solution of hard computational problems, Science 268 (1995) 542–545.
[4] Q. Quyang, P.D. Kaplan, S. Liu, A. Libchaber, DNA solution of the maximal clique problem, Science 278 (1997) 446–449.
[5] M. Arita, A. Suyama, M. Hagiya, A heuristic approach for Hamiltonian path problem with molecules, in: Proceedings of the Second Genetic Programming (GP-97), 1997, pp. 457–462.
[6] N. Morimoto, M. Arita, A. Suyama, Solid phase DNA solution to the Hamiltonian path problem, DIMACS (Series in Discrete Mathematics and Theoretical Computer Science), vol. 48, 1999, pp. 93–206.
[7] A. Narayanan, S. Zorbala, DNA algorithms for computing shortest paths, in: J.R. Koza, et al. (Eds.), Proceedings of the Third Annual Conference on Genetic Programming 1998, 1998, pp. 718–724.

[8] S.-Y. Shin, B.-T. Zhang, S.-S. Jun, Solving traveling salesman problems using molecular programming, in: Proceedings of the 1999 Congress on Evolutionary Computation (CEC99), vol. 2, 1999, pp. 994–1000.
[9] T.H. Cormen, C.E. Leiserson, R.L. Rivest, Introduction to Algorithms.
[10] M.R. Garey, D.S. Johnson, Computer and Intractability, Freeman, San Francisco, CA, 1979.
[11] D. Boneh, C. Dunworth, R.J. Lipton, J. Sgall, On the computational power of DNA, Discrete Applied Mathematics, Special Issue on Computational Molecular Biology 71 (1996) 79–94.
[12] L.M. Adleman, On constructing a molecular computer, in: R. Lipton, E. Baum (Eds.), DNA Based Computers, DIMACS: Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, 1996, pp. 1–21.
[13] M. Amos, DNA computation, Ph.D. Thesis, Department of Computer Science, The University of Warwick, 1997.
[14] S. Roweis, E. Winfree, R. Burgoyne, N.V. Chelyapov, M.F. Goodman, P.W.K. Rothemund, L.M. Adleman, A sticker based model for DNA computation, in: L. Landweber, E. Baum (Eds.), Proceedings of the Second Annual Workshop on DNA Computing, Princeton University, DIMACS: Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, 1999, pp. 1–29.
[15] M.J. Perez-Jimenez, F. Sancho-Caparrini, 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, 2001.
[16] G. Paun, G. Rozenberg, A. Salomaa, DNA Computing: New Computing Paradigms. Springer-Verlag, New York, 1998. ISBN 3-540-64196-3.
[17] W.-L. Chang, M. Guo, 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, 2002, pp. 167–172.
[18] W.-L. Chang, M. Guo, Solving the clique problem and the vertex cover problem in Adleman–Lipton's model, IASTED International Conference, Networks, Parallel and Distributed Processing, and Applications, Japan, 2002, pp. 431–436.
[19] W.-L. Chang, M. Guo, Solving NP-complete problem in the Adleman–Lipton model, in: Proceedings of the 2002 International Conference on Computer and Information Technology, Japan, 2002, pp. 157–162.
[20] W.-L. Chang, M. Guo, Resolving the 3-dimensional matching problem and the set packing problem in Adleman–Lipton's model, IASTED International Conference, Networks, Parallel and Distributed Processing, and Applications, Japan, 2002, pp. 455–460.
[21] B. Fu, Volume bounded molecular computation, Ph.D. Thesis, Department of Computer Science, Yale University, 1997.
[22] R.S. Braich, C. Johnson, P.W.K. Rothemund, D. Hwang, N. Chelyapov, L.M. Adleman, 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.
[23] K. Mir, A restricted genetic alphabet for DNA computing, in: E.B. Baum, L.F. Landweber (Eds.), DNA Based Computers II: DIMACS Workshop, vol. 44, June 10–12, 1996, DIMACS: Series in Discrete Mathematics and Theoretical Computer Science, Providence, RI, 1998, pp. 243–246.
[24] A.R. Cukras, D. Faulhammer, R.J. Lipton, L.F. Landweber, Chess games: a model for RNA-based computation, in: Proceedings of the Fourth DIMACS Meeting on DNA Based Computers, held at the University of Pennsylvania, June 16–19, 1998, pp. 27–37.
[25] M.S.-H. Ho, M. Guo, W.-L. Chang, Using sticker for solving the dominating-set problem in the Adleman–Lipton model, IEICE Transaction on Information and Systems, submitted for publication.
[26] M.S.-H. Ho, W.-L. Chang, M. Guo, Solving the set-basis problem in sticker-based model and the Adleman–Lipton model, ICPP-HPSECA03, Kaohsiung, Taiwan, Republic of China, in press.

Weng-Long Chang was born on March 26, 1966 in Tainan County, Taiwan, Republic of China. He received the BS degree in Computer Science and Information Engineering from Feng Chia University in 1988. He obtained the MS and PhD degrees in Computer Science and Information Engineering from National Cheng Kung University, Taiwan, in 1994 and 1999, respectively. His research interests include molecular computing, and languages and compilers for parallel computing.

Minyi Guo received his PhD 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 School of Computer Science and Engineering, 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 chair 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, IEEE Computer Society, and IEICE. He is listed in Marquis Who's Who in Science and Engineering.

Michael Ho, Assistant 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 such as the GAS Company, Fox, Fidelity, ARCO, US LA Hall Records, ... etc. He is also certified in Oracle DBA/Developer (OCP), CCNA/CCNP, MCSE, Unix Solaris systems SA, and Dell server/computer technician engineer. 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 PhD in IS/CS with Management and Accounting minor from the University of Texas at Austin and a MS 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.