Available online at www.sciencedirect.com
SCIENCE DIRECT®
BioSystems 73 (2004) 117–130
BioSystems
www.elsevier.com/locate/biosystems

Molecular solutions for the subset-sum problem on DNA-based supercomputing

Weng-Long Chang [a,1], Michael (Shan-Hui) Ho [a,1], Minyi Guo [b,*]

[a] *Department of Information Management, Southern Taiwan University of Technology, Tainan County 710, Taiwan, ROC*
[b] *Department of Computer Software, The University of Aizu, Aizu-Wakamatsu City, Fukushima 965-8580, Japan*

Received 28 July 2003; received in revised form 19 November 2003; accepted 20 November 2003

Abstract

In this paper our main purpose is to give *molecular* solutions for the subset-sum problem. In order to achieve this, we propose a DNA-based algorithm of an *n*-bit parallel adder and a DNA-based algorithm of an *n*-bit parallel comparator to formally verify our designed molecular solutions for the subset-sum problem.

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

1. Introduction

Through advances in molecular biology (Sinden, 1994), it is now possible to produce roughly 10^{18} DNA strands that fit in a test tube. Those 10^{18} DNA strands can also be applied for representing 10^{18} bits of information. Basic biological operations can be used to simultaneously operate 10^{18} bits of information. Or we can say that 10^{18} data processors can be executed in parallel. Hence, it becomes obvious that biological computing can provide a huge parallelism for dealing with problems in the real world.

Feynman first proposed molecular computation in 1961, but his idea was not implemented by experiment for a few decades. In Adleman (1994) succeeded to solve an instance of the Hamiltonian path problem in a test tube, just by handling DNA strands. Lipton (1995) demonstrated that the Adleman techniques could be used to solve the satisfiability problem (the first NP-complete problem). Adleman and co-authors (Roweis et al., 1999) proposed *sticker* for enhancing the Adleman–Lipton model.

In this paper, first we use *sticker* to construct solution spaces of DNA strands for the *subset-sum* problem. Then by using biological operations in the Adleman–Lipton model, we develop a DNA-based algorithm of parallel adder and a DNA-based algorithm of parallel comparative operator for finishing the functions of add and comparative instructions. We also show that the subset-sum problem is solved by using the biological operations in the Adleman–Lipton model for the sticker solution space. Furthermore, this work presents clear evidence of the ability of molecular computing to solve the NP-complete problem with mathematical operations.

The paper is organized as follows. Section 2 introduces the Adleman–Lipton model in detail then this model is compared with other models. Section 3

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

0303-2647/$ – see front matter
doi:10.1016/j.biosystems.2003.11.001

introduces the DNA program to solve the subset-sum problem for the sticker solution space. In Section 4, the experimental results by simulated DNA computing are given. Conclusions and future research work are drawn in Section 5.

2. DNA model of computation

2.1. The Adleman–Lipton model

A deoxyribonucleic acid (DNA) 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. These bases are 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, we can 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. (For more discussions of the relevant biological background refer to Sinden, 1994; Boneh et al., 1996; Paun et al., 1998.)

In the Adleman–Lipton model (Adleman, 1994; Lipton, 1995), *splints* were used to construct the corresponding edges of a particular graph of paths, which represented all possible binary numbers. As it stands, their construction indiscriminately builds all splints that lead to a complete graph. This is to say that hybridization has a higher probability of errors. Hence, Adleman and co-authors (Roweis et al., 1999) 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.

The DNA operations in the Adleman–Lipton model (Adleman, 1994, 1996; Lipton, 1995; Boneh et al., 1996) are described below. These operations will be used for finding solutions of the subset-sum problem.

2.1.1. 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 $\{A, C, G, T\}$). A simple notation is used to explain the various operations to be performed on DNA. Given a string X over the alphabet $\{A, C, G, T\}$, $\uparrow X$ is denoted as the single stranded DNA which is made up of the letters of X oriented from the 5′ end to the 3′ end (the first letter of X is on the 5′ end). $\downarrow X$ is denoted as the Watson–Crick complement of the strand $\uparrow X$. When $\downarrow X$ and $\uparrow X$ anneal to each other they form a double strand which we denote by $\updownarrow X$. If given a tube, one can perform the following operations:

1. *Extract*. Given a tube P and a short single strand of DNA, S, produces 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, without any change in the individual strands.
3. *Detect*. Given a tube P, if P includes at least one DNA molecule we have 'yes,' and if P contains no DNA molecule we have 'no.'
4. *Discard*. Given a tube P, the operation will discard the tube P.
5. *Amplify*. At times we need to make copies of all the DNA strands in a test tube. This can be done with a straightforward application of the polymerase chain reaction (PCR). PCR is a process that uses DNA polymerase to make many copies of a DNA sequence. PCR works in the following way. If we have the duplex $\updownarrow XYZ$, we first melt it to form $\uparrow XYZ$ and $\downarrow XYZ$. To this solution we add the primer oligos $\downarrow Z$ and $\uparrow X$, which anneals to form the partial duplexes $\uparrow XY\updownarrow Z$ and $\updownarrow X\downarrow YZ$. DNA polymerase can then elongate the primers to create full duplexes of the form $\updownarrow XYZ$. Note that we now have two copies of our original strand. If we just repeat this process, we will again double the number of copies of the original strand in solution. Soon we will have 4 copies, then 8, then 16, and so on, until we have enough copies for our purposes. Thus, if we can guarantee that the primer sequences that we use occurs on the ends

of every strand, and only on the ends, then we can use PCR to duplicate every strand in the test tube. We call this operation *Amplify*. Given a tube P, the operation, *Amplify*(P, P_1, P_2) will produce two new tubes $P_1$and P_2 such that tube P_1 is a copy of tube P, tube P_2 is also a copy of tube P (which are now identical) and tube P becomes empty tube.

6. *Append*. Given a tube P containing 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. *Read*. Given a tube P, the operation is used to describe a single molecule, which is contained in tube P. Even if P contains many different molecules 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,d) 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. The methods used for resolving problems have exponentially increased volumes of DNA and linearly increased the time.

Bach et al. (1996) proposed a $n1.89^n$ volume, $O(n^2 + m^2)$ time molecular algorithm for the 3-coloring problem and a 1.51^n volume, $O(n^2m^2)$ 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 is higher.

Quyang et al. (1997) showed that enzymes could be used to solve the NP-complete clique problem. Because the maximum number of vertices that they can process is limited to 27, the maximum number of DNA strands for solving this problem is 2^{27} (Quyang et al., 1997). Shin et al. (1999) presented an encoding scheme for decreasing the error rate of hybridization. This method (Shin et al., 1999) can be employed towards ascertaining the traveling salesman problem for representing integers and real values with fixed-length codes. Arita et al. (1997) and Morimoto et al. (1999) proposed a new molecular experimental technique and a solid-phase method to find a Hamiltonian path. Amos (1997) proposed a parallel filtering model for resolving the Hamiltonian path problem, the sub-graph isomorphism problem, the 3-vertex-colorability problem, the clique problem and the independent-set problem. These methods (Arita et al., 1997; Morimoto et al., 1999; Amos, 1997) have lowered the error rate in real molecular experiments.

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

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

3. Molecular solutions for the subset-sum problem

3.1. Definition of the subset-sum problem

Assume that a finite set S is $\{s_1, \ldots, s_q\}$, where s_m is the mth element for $1 \leq m \leq q$. Also suppose that every element in S is a positive integer. Assume that

$|S|$ is the number of elements in S and $|S|$ is equal to q. The subset-sum problem for S is to find a subset $S^1 \subseteq S$ such that the sum of every element in S^1 is exactly b, where b is a positive integer. The subset-sum problem has been proved to be the NP-complete problem (Cormen et al., 2003; Garey and Johnson, 1979; Cook, 1971; Karp, 1972).

In Eq. (1), a finite set S is $\{1, 2\}$ and any given positive integer b is 3. Four subsets of S are, respectively, $\varnothing$, $\{1\}$, $\{2\}$ and $\{1, 2\}$. Since the sum of the first element and the second element in $\{1, 2\}$ is equal to 3, the solution of the subset-sum problem for S is $\{1, 2\}$.

$$S = \{1, 2\} \text{ and } b = 3 \tag{1}$$

The above equation contains finite set and the given positive integer of our problem.

3.2. Sticker-based solution space for subsets of a finite set

Assume that $x_1, \ldots, x_q$ is a q-bit binary number, which is applied to represent q elements in a finite set S. *Sticker* in a sticker-based model (Roweis et al., 1999; Braich et al., 2003) is a 15-base value sequence. For every bit x_m representing the mth element in S to $1 \leq m \leq q$, two *distinct* 15-base value sequences were designed. One represents the value "0" for x_m and the other represents the value "1" for x_m. For the sake of convenience in our presentation, assume that x_m^1 denotes the value of x_m to be 1 and x_m^0 defines the value of x_m to be 0. The following algorithm is used to construct sticker-based solution space for 2^q possible subsets of a q-element set S.

Procedure Init(T_0, q)
  (1) For $m = 1$ to q
    (1a) Amplify(T_0, T_1, T_2).
    (1b) Append(T_1, x_m^1).
    (1c) Append(T_2, x_m^0).
    (1d) $T_0 = \cup(T_1, T_2)$.
  EndFor
EndProcedure

Lemma 1. *Sticker-based solution space of 2^q possible subsets for a q-element set S can be constructed with sticker in a sticker-based model from the algorithm,* Init(T_0, q).

Proof. Assume that T_0, T_1 and T_2 are distinct test tubes but only T_1 and T_2 are empty. The algorithm, Init(T_0, q), is implemented via the *amplify*, *append* and *merge* operations. Each time Step 1(a) is used to amplify tube T_0 and to generate two new tubes, T_1 and T_2, which are copies of T_0. Tube T_0 becomes empty. Then, Step 1(b) is applied to append a DNA sequence (sticker), representing the value "1" for x_m, onto the end of every strand in tube T_1. This is to say that subsets containing the mth element appear in tube T_1. Step 1(c) is also employed to append a DNA sequence (sticker), representing the value "0" for x_m, onto the end of every strand in tube T_2. That implies that subsets not containing the mth element appear in tube T_2. Next, Step 1(d) is used to pour tube T_1 and T_2 into tube T_0. This indicates that DNA strands in tube T_0 include DNA sequences of $x_m = 1$ and $x_m = 0$. After repeating execution of Steps 1(a)–1(d), it finally produces tube T_0 that consists of 2^q DNA sequences representing 2^q possible subsets. Therefore, it is inferred that sticker-based solution space for 2^q possible subsets of a q-element set S can be constructed with sticker.

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

3.3. Sticker-based solution space for elements of subsets for a finite set

An element, s_m, in a q-element finite set S can be converted as a binary number, $s_{m,1}, \ldots, s_{m,n}$, using a digital computer. $s_{m,n}$ is the highest order bit and $s_{m,1}$ is the lowest order bit. *Sticker* is applied to represent every bit $s_{m,k}$ for $1 \leq k \leq n$. For every bit $s_{m,k}$ two *distinct* DNA sequences were designed. One corresponds to the value "0" for $s_{m,k}$ and the other corresponds to the value "1" for $s_{m,k}$. For the sake of convenience in our presentation, assume that $s_{m,k}^1$ denotes the value of $s_{m,k}$ to be 1 and $s_{m,k}^0$ defines the value of $s_{m,k}$ to be 0. The following algorithm is employed to construct sticker-based solution space for elements of 2^q possible subsets to a q-element set S.

Procedure Value(T_0, q, n)
(1) For $m = 1$ to q
(1a) $T_1 = +(T_0, x_m^1)$ and $T_2 = -(T_0, x_m^1)$.
(1b) For $k = 1$ to n
(1c) Append(T_1, $s_{m,k}$).
(1d) Append(T_2, $s_{m,k}^0$).
EndFor
(1e) $T_0 = \cup(T_1, T_2)$.
EndFor
EndProcedure

Lemma 2. *Sticker-based solution space of elements for 2^q possible subsets of a q-element set S can be constructed with sticker in a sticker-based model from the algorithm,* Value(T_0, q, n).

Proof. Assume that T_0 is generated from the algorithm, Init(T_0, q), and contains those DNA strands corresponding to 2^q possible subsets to a q-element set S. The algorithm, Value(T_0, q, n), is implemented via the *extract*, *append* and *merge* operations. Step 1 is the outer loop and is employed to construct sticker-based solution space. When the first execution of Step 1(a) uses the *extract* operation to form two test tubes: T_1 and T_2. The first tube T_1 includes all of the strands that have $x_1 = 1$, that is to say, the first element in S occurs in tube T_1. The second tube T_2 consists of all of the strands that have $x_1 = 0$, this is to say that the first element in S does not appear in tube T_2. Step 1(b) is the inner loop and is mainly used to encode the size of an element onto the tail of those DNA strands containing the element. On the first execution of Step 1(c), it uses the *append* operation to append 15-based DNA sequences for representing the value 1 of $s_{1,1}$ or representing the value 0 of $s_{1,1}$ onto the tail of every strand in T_1. Next, the first execution of Step 1(d) also applies the *append* operation to append 15-based DNA sequences for representing the value 0 of $s_{1,1}$ onto the tail of every strand in T_2. Repeat execution of Steps 1(c)–1(d) until every bit in the first element in S is processed. Tube T_1 contains the strands encoding the size of the first element. Tube T_2 includes the strands encoding the value 0 of $s_{1,k}$ for $1 \le k \le n$. Then Step 1(e) uses the *merge* operation to pour two tubes T_1 and T_2 into tube T_0. Tube T_0 currently consists of the DNA sequences encoding the size of the first element and the value 0 of $s_{1,k}$ for $1 \le k \le n$. Similarly, after other elements are processed, tube T_0 consists of the strands for encoding elements of 2^q possible subsets for a q-element set S.

From Value(T_0, q, n), it takes q *extract* operations, $2 \times n \times q$ *append* operations, q *merge* operations and three test tubes to construct sticker-based solution space for elements of 2^q possible subsets to a q-element set S. A q-bit binary number corresponds to a subset and an n-bit binary number encodes the size of an element in S. A value sequence for every bit contains 15 bases. Therefore, the length of a DNA strand, encoding elements of 2^q possible subsets for a q-element set S, is $15 \times (q+n)$ bases consisting of the concatenation of one value sequence for each bit. □

3.4. Parallel adder for computing the sum of elements

Assume that $((q+1) \times n)$ one-digit binary numbers, y_w, are used to represent the sum of q elements in S for $1 \le w \le ((q+1) \times n)$. Also suppose $(q \times (n+1))$ one-digit binary numbers, z_a, are applied to represent the carry of the sum for q elements of S to $1 \le a \le (q \times (n+1))$. Assume that $y_1 \ldots y_n$ is employed to denote the initialized value of the sum for finishing the addition of every element in S. Also suppose that $y_{(i-1)\times n+j} \ldots y_{i\times n}$ is applied to denote the intermediate value of the sum to finish the addition of every element in S for $2 \le i \le q$ and $1 \le j \le n$. Assume that $y_{q\times n+j} \ldots y_{(q+1)\times n}$ is employed to denote the final result of the sum to finish the addition of every element in S for $1 \le j \le n$. Also suppose that $z_{(b-1)\times(n+1)+1}$ is used to define the initialized value of the carry for finishing the addition of every element in S for $1 \le b \le q$. Assume that $z_{(b-1)\times(n+1)+c} \ldots z_{b\times(n+1)}$ is employed to define the intermediate value of the carry to finish the addition of every element in S for $1 \le b \le q-1$ and $2 \le c \le n+1$. Also suppose that $z_{(q-1)\times(n+1)+c} \ldots z_{q\times(n+1)}$ is used to define the final result of the carry for finishing the addition of every element in S to $2 \le c \le n+1$.

Sticker is employed to encode every bit y_w and z_a for $1 \le w \le ((q+1) \times n)$ and $1 \le a \le (q \times (n+1))$. For every bit y_w two *distinct* DNA sequences were designed. One corresponds to the value "0" for y_w and the other corresponds to the value "1" for y_w. Similarly, there also are two *distinct* DNA sequences designed for every bit z_a. One corresponds to the value "0" for z_a and the other corresponds to the value "1" for z_a. For the sake of convenience in our presentation,

assume that y_w^1 denotes the value of y_w to be 1 and y_w^0 defines the value of y_w to be 0. Similarly, suppose that z_a^1 contains the value of z_a to be 1 and z_a^0 contains the value of z_a to be 0. The following algorithm is proposed to finish the function of a parallel adder for computing the sum of elements to be 2^q possible subsets of a q-element set S.

Procedure ParallelAdder(T_0, q, n)
(1) For $j = 1$ to n
(1a) Append(T_0, y_j^0).
EndFor
(2) For $i = 1$ to q
(2a) Append(T_0, $z^0_{(i-1)\times(n+1)+1}$).
For $j = 1$ to n
(2b) $T_1 = +(T_0, s^1_{i,j})$ and $T_2 = -(T_0, s^1_{i,j})$.
(2c) $T_3 = +(T_1, z^1_{(i-1)\times(n+1)+j})$ and $T_4 = -(T_1, z^1_{(i-1)\times(n+1)+j})$.
(2d) $T_5 = +(T_2, z^1_{(i-1)\times(n+1)+j})$ and $T_6 = -(T_2, z^1_{(i-1)\times(n+1)+j})$.
(2e) $T_7 = +(T_3, y^1_{(i-1)\times n+j})$ and $T_8 = -(T_3, y^1_{(i-1)\times n+j})$.
(2f) $T_9 = +(T_4, y^1_{(i-1)\times n+j})$ and $T_{10} = -(T_4, y^1_{(i-1)\times n+j})$.
(2g) $T_{11} = +(T_5, y^1_{(i-1)\times n+j})$ and $T_{12} = -(T_5, y^1_{(i-1)\times n+j})$.
(2h) $T_{13} = +(T_6, y^1_{(i-1)\times n+j})$ and $T_{14} = -(T_6, y^1_{(i-1)\times n+j})$.
(2i) Append(T_7, $y^1_{i\times n+j}$) and Append(T_7, $z^1_{(i-1)\times(n+1)+(j+1)}$).
(2j) Append(T_8, $y^0_{i\times n+j}$) and Append(T_8, $z^1_{(i-1)\times(n+1)+(j+1)}$).
(2k) Append(T_9, $y^0_{i\times n+j}$) and Append(T_9, $z^1_{(i-1)\times(n+1)+(j+1)}$).
(2l) Append(T_{10}, $y^1_{i\times n+j}$) and Append(T_{10}, $z^0_{(i-1)\times(n+1)+(j+1)}$).
(2m) Append(T_{11}, $y^0_{i\times n+j}$) and Append(T_{11}, $z^1_{(i-1)\times(n+1)+(j+1)}$).
(2n) Append(T_{12}, $y^1_{i\times n+j}$) and Append(T_{12}, $z^0_{(i-1)\times(n+1)+(j+1)}$).
(2o) Append(T_{13}, $y^1_{i\times n+j}$) and Append(T_{13}, $z^0_{(i-1)\times(n+1)+(j+1)}$).
(2p) Append(T_{14}, $y^0_{i\times n+j}$) and Append(T_{14}, $z^0_{(i-1)\times(n+1)+(j+1)}$).
(2q) $T_0 = \cup(T_7, T_8, T_9, T_{10}, T_{11}, T_{12}, T_{13}, T_{14})$.
EndFor
EndFor
EndProcedure

Lemma 3. *The algorithm,* ParallelAdder(T_0, q, n) *can be applied to finish the function of a parallel adder for computing the sum of elements to* 2^q *possible subsets of a* q*-element set* S.

Proof. Assume that T_0 is generated from the algorithm, Value(T_0, q, n), and contains the DNA strands corresponding to the elements of 2^q possible subsets for q-element set S. The algorithm, ParallelAdder(T_0, q, n), is implemented via the *extract*, *append* and *merge* operations. Step 1 is the first loop and is mainly employed to set the initialized value of the sum for q elements in S. When the first execution of Step 1(a) uses the *append* operation to append 15-based DNA sequences for representing y_1^0 onto the tail of every strand in T_0. This is to say that the first bit of the initialized value for the sum is set to zero. Repeating execution of Step 1(a) until every bit of the initialized value for the sum is processed. Tube T_0 contains the strands encoding the initialized value of the sum.

Step 2 is the second loop and is mainly used to finish the function of a parallel adder. On the first execution of Step 2(a), it uses the *append* operation to append 15-based DNA sequences for encoding z_1^0 onto the tail of every strand in T_0. This implies that the first bit of the initialized value for the carry is also set to zero. When the first execution of Step 2(b) employs the *extract* operation to form two test tubes: T_1 and T_2. The first tube T_1 includes all of the strands that have $s_{1,1} = 1$. The second tube T_2 consists of all of the strands that have $s_{1,1} = 0$. On the first execution of Step 2(c), it uses the *extract* operation to form two test tubes: T_3 and T_4. The first tube T_3 includes all of the strands that have $s_{1,1} = 1$ and $z_1 = 1$. The second tube T_4 consists of all of the strands that have $s_{1,1} = 1$ and $z_1 = 0$. Next, the first execution of Step 2(d) uses the *extract* operation to form two test tubes: T_5 and T_6. The first tube T_5 includes all of the strands that have $s_{1,1} = 0$ and $z_1 = 1$. The second tube T_6 consists of

all of the strands that have $s_{1,1} = 0$ and $z_1 = 0$. On the first execution of Step 2(e), it uses the *extract* operation to form two test tubes: T_7 and T_8. The first tube T_7 includes all of the strands that have $s_{1,1} = 1$, $z_1 = 1$ and $y_1 = 1$. The second tube T_8 consists of all of the strands that have $s_{1,1} = 1$, $z_1 = 1$ and $y_1 = 0$. Then, on the first execution of Step 2(f), it applies the *extract* operation to form two test tubes: T_9 and T_{10}. The first tube T_9 includes all of the strands that have $s_{1,1} = 1$, $z_1 = 0$ and $y_1 = 1$. The second tube T_{10} consists of all of the strands that have $s_{1,1} = 1$, $z_1 = 0$ and $y_1 = 0$. On the first execution of Step 2(g), it employs the *extract* operation to form two test tubes: T_{11} and T_{12}. The first tube T_{11} includes all of the strands that have $s_{1,1} = 0$, $z_1 = 1$ and $y_1 = 1$. The second tube T_{12} consists of all of the strands that have $s_{1,1} = 0$, $z_1 = 1$ and $y_1 = 0$. Next, on the first execution of Step 2(h) uses the *extract* operation to form two test tubes: T_{13} and T_{14}. The first tube T_{13} includes all of the strands that have $s_{1,1} = 0$, $z_1 = 0$ and $y_1 = 1$. The second tube T_{14} consists of all of the strands that have $s_{1,1} = 0$, $z_1 = 0$ and $y_1 = 0$.

Next, the first execution of Step 2(i) uses the *append* operations to append y_{n+1}^1 and z_2^1 onto the tail of every strand in T_7. On the first execution of Step 2(j), it applies the *append* operations to append y_{n+1}^0 and z_2^1 onto the tail of every strand in T_8. Then, the first execution of Step 2(k) employs the *append* operations to append y_{n+1}^0 and z_2^1 onto the tail of every strand in T_9. On the first execution of Step 2(l), it uses the *append* operations to append y_{n+1}^1 and z_2^0 onto the tail of every strand in T_{10}. Next, the first execution of Step 2(m) uses the *append* operations to append y_{n+1}^0 and z_2^1 onto the tail of every strand in T_{11}. On the first execution of Step 2(n), it uses the *append* operations to append y_{n+1}^1 and z_2^0 onto the tail of every strand in T_{12}. Then, the first execution of Step 2(o) applies the *append* operations to append y_{n+1}^1 and z_2^0 onto the tail of every strand in T_{13}. On the first execution of Step 2(p), it employs the *append* operations to append y_{n+1}^0 and z_2^0 onto the tail of every strand in T_{14}. Next, the first execution of Step 2(q) applies the *merge* operation to pour tubes T_7 through T_{14} into T_0. Repeat execution of Steps 2(a)–2(q) until every bit of q elements in S is processed. Tube T_0 contains the strands encoding the sum of q elements in S.

From ParallelAdder(T_0, q, n), it takes $(7 \times n \times q)$ *extract* operations, $(n + q + 16 \times n \times q)$ *append* operations, $(n \times q)$ *merge* operations and 15 test tubes to compute the sum of elements for 2^q possible subsets of a q-element set S. A q-bit binary number corresponds to a subset. An n-bit binary number encodes the size of an element in S. $((q + 1) \times n)$ bits correspond to the sum of elements. $(q \times (n + 1))$ bits encode the carry of the sum. A value sequence for every bit contains 15 bases. Therefore, the length of a DNA strand, encoding the final result of the sum for elements of 2^q possible subsets to a q-element set S, is $15 \times (q + n + (q + 1) \times n + q \times (n + 1))$ bases consisting of the concatenation of one value sequence for each bit. □

3.5. Parallel comparator for comparing the sum of elements for subsets of a finite set with any given positive integer

Any given positive integer, b, can be represented as n one-bit binary numbers, $y_{q\times n+j}$ for $1 \le j \le n$, using a digit computer. This is to say that the same sticker is used to encode b and the final result of the sum for elements of 2^q possible subsets to a q-element set S. The main advantage is to reduce the complexity of the DNA algorithm for constructing a parallel comparator. MakeValue(T_b, q, n) is proposed to construct a DNA strand for encoding b. OneBitComparator(T_0, T_b, q, n, j) is presented to finish the function of a parallel comparator for one bit and ParallelComparator(T_0, q, n) also is proposed to finish the function of a parallel comparator for comparing the sum of elements to 2^q possible subsets of a q-element set S with b.

Procedure MakeValue(T_b, q, n)
(1) For $j = 1$ to n
(1a) Append(T_b, $y_{q\times n+j}$).
EndFor

Lemma 4. *Sticker-based solution space of any given positive integer b can be constructed with sticker in a sticker-based model from the algorithm,* MakeValue(T_b, q, n).

Proof. The algorithm, MakeValue(T_b, q, n), is implemented via the *append* operation. The only loop in the algorithm is mainly used to construct a DNA strand

for any given positive integer b. Each time Step 1(a) is used to append a DNA sequence (sticker), encoding the value "1" or "0" of $y_{q\times n+j}$, onto the end of every strand in tube T_b. After repeating execution of Step 1(a), it finally produces tube T_b that includes a DNA strand encoding b. Therefore, it is inferred that sticker-based solution space for any given positive integer b can be constructed with sticker.

From MakeValue(T_b, q, n), it takes n *append* operations and one test tube to construct sticker-based solution space. Any given positive integer b corresponds to n one-bit binary numbers. A value sequence for every bit contains 15 bases. Therefore, the length of a DNA strand, encoding b, is $15 \times n$ bases consisting of the concatenation of one value sequence for each bit. □

Procedure OneBitComparator(T_0, T_b, q, n, j)
(1) $T_0^{\mathrm{ON}} = +(T_0, y^1_{q\times n+j})$ and $T_0^{\mathrm{OFF}} = -(T_0, y^1_{q\times n+j})$.
(2) $T_b^{\mathrm{ON}} = +(T_b, y^1_{q\times n+j})$ and $T_b^{\mathrm{OFF}} = -(T_b, y^1_{q\times n+j})$.
(3) If (Detect(T_b^{ON}) = '*yes*') then
(3a) If (Detect(T_0^{ON}) = '*yes*') then
(3b) $T_0 = \cup(T_0, T_0^{\mathrm{ON}})$ and $T_b = \cup(T_b, T_b^{\mathrm{ON}})$.
EndIf
Else
(3c) If (Detect(T_0^{OFF}) = '*yes*') then
(3d) $T_0 = \cup(T_0, T_0^{\mathrm{OFF}})$ and $T_b = \cup(T_b, T_b^{\mathrm{OFF}})$.
EndIf
EndIf
EndProcedure

Lemma 5. *The algorithm,* OneBitComparator(T_0, T_b, q, n, j) *can be applied to finish the function of a one-bit parallel comparator*.

Proof. The algorithm, OneBitComparator(T_0, T_b, q, n, j), is implemented via the *extract*, *detect* and *merge* operations. When the execution of Step 1 employs the *extract* operation to form two test tubes: T_0^{ON} and T_0^{OFF}. The first tube T_0^{ON} includes all of the strands that have $y_{q\times n+j} = 1$. The second tube T_0^{OFF} consists of all of the strands that have $y_{q\times n+j} = 0$. On the execution of Step 2, it also uses the *extract* operation to form two test tubes: T_b^{ON} and T_b^{OFF}. The first tube T_b^{ON} includes all of the strands that have $y_{q\times n+j} = 1$. The second tube T_b^{OFF} consists of all of the strands that have $y_{q\times n+j} = 0$. Next, the execution of Step 3 uses the *detect* operation to check whether there is any DNA sequence in tube T_b^{ON}. If it returns a "yes," this indicates that the value of the jth bit in b is one. On the execution of Step 3(a), it uses the *detect* operation to test if there is any DNA sequence in tube T_0^{ON}. If it returns a "yes," this indicates that the value of the jth bit in the sum of q elements in S is also one. Next, the execution of Step 3(b) applies the *merge* operations to pour tubes T_0^{ON} into T_0 and also to pour tubes T_b^{ON} into T_b. If the *detect* operation in Step 3 returns a "no," the execution of Step 3(c) uses the *detect* operation to check whether there is any DNA strand in tube T_0^{OFF}. If it returns a "yes," this indicates that the value of the jth bit in the sum of q elements in S is zero. Then, the execution of Step 3(d) applies the *merge* operations to pour tubes T_0^{OFF} into T_0 and also to pour tubes T_b^{OFF} into T_b.

From OneBitComparator(T_0, T_b, q, n, j), it takes two *extract* operations, three *detect* operations, four *merge* operations and six test tubes to finish the function of a one-bit parallel comparator. □

Procedure ParallelComparator(T_0, T_b, q, n)
(1) For $j = n$ to 1
(1a) OneBitComparator(T_0, T_b, q, n, j).
(1b) If (Detect(T_0)="no") or (Detect(T_b)="no") then
(1c) Terminate the execution of the loop.
EndIf
EndFor
EndProcedure

Lemma 6. *The algorithm,* ParallelComparator(T_0, T_b, q, n) *can be used to finish the function of an n-bit parallel comparator*.

Proof. The only loop is used to implement the function of an n-bit parallel comparator. When the first execution of Step 1(a) calls the algorithm, OneBitComparator(T_0, T_b, q, n, j), to finish the comparative result of the highest order bit for any given positive integer b and the sum of q elements in S. On the first execution of Step 1(b), it uses the *detect* operations to check whether there is any DNA

sequence in T_0 or T_b. From OneBitComparator(T_0, T_b, q, n, j), if the two highest order bits are not equal, then T_0 and T_b are both empty. This means that the execution of Step 1(c) will terminate the execution of the loop if the two compared bits are not equal. Otherwise it repeats execution of Steps 1(a)–1(c) until every bit of the sum for q elements in S is compared with every bit of b. Tube T_0 contains the strands encoding the sum of q elements, which is equal to b.

From ParallelComparator(T_0, T_b, q, n), it takes $2 \times n$ *extract* operations, $5 \times n$ *detect* operations, $4 \times n$ *merge* operations and six test tubes to finish the function of an n-bit parallel comparator. □

3.6. A DNA algorithm for solving the subset-sum problem

Algorithm 1. Solving the subset-sum problem.

(1) Init(T_0, q).
(2) Value(T_0, q, n).
(3) ParallelAdder(T_0, q, n).
(4) MakeValue(T_b, q, n).
(5) ParallelComparator(T_0, T_b, q, n).
(6) If (Detect(T_0) = "yes") then
  (6a) Read(T_0).
EndIf
EndAlgorithm

Theorem 1. *From those steps in* Algorithm 1, *the subset-sum problem for* 2^q *possible subsets of a q-element set S can be solved.*

Proof. On the execution of Step 1, it calls Init(T_0, q). The algorithm, Init(T_0, q), is mainly used to construct sticker-based solution space for 2^q possible subsets of a q-element set S. This means that tube T_0 includes strands encoding 2^q possible subsets. Next, the execution of Step 2 calls Value(T_0, q, n). The algorithm, Value(T_0, q, n), is employed to construct sticker-based solution space for elements of 2^q possible subsets. This implies that tube T_0 contains strands encoding elements of 2^q possible subsets. On the execution of Step 3, it calls ParallelAdder(T_0, q, n). The algorithm, ParallelAdder(T_0, q, n), is used to finish the function of a parallel adder for computing the sum of elements to 2^q possible subsets. That means tube T_0 consists of strands encoding the sum of elements for 2^q possible subsets. Next, the execution of Step 4 calls MakeValue(T_b, q, n). The algorithm, MakeValue(T_b, q, n), is applied to construct a DNA strand for encoding b. This indicates that tube T_b consists of a strand encoding b. On the execution of Step 5, it calls ParallelComparator(T_0, T_b, q, n). The algorithm, ParallelComparator(T_0, T_b, q, n), is employed to finish the function of a parallel comparator for comparing the sum of elements with b. Next, the execution of Step 6 uses the *detect* operation to check if there is any DNA strand in tube T_0. If it returns a "yes," the execution of Step 6(a) applies the *read* operation to find the answer. Therefore, the subset-sum problem for 2^q possible subsets of a q-element set S can be solved from those steps in Algorithm 1.

The set S and the integer b in Eq. (1) can be applied for showing the power of Algorithm 1. In Eq. (1), the finite set S is $\{1, 2\}$ and the integer b is 3. Four subsets of S are, respectively, $\varnothing$, $\{1\}$, $\{2\}$ and $\{1, 2\}$. From the execution of Step 1 in Algorithm 1, tube T_0 is filled with four library strands. The four library strands are $0(x_1 = 0)0(x_2 = 0), 1(x_1 = 1)0(x_2 = 0), 0(x_1 = 0)1(x_2 = 1)$ and $1(x_1 = 1)1(x_2 = 1)$. They encode, respectively to, $\varnothing$, $\{1\}$, $\{2\}$ and $\{1, 2\}$. Next, from the execution of Step 2 in Algorithm 1, DNA sequences encoding the size of the two elements in S are appended onto the tail of the four library strands in tube T_0. The library strand, $0(x_1 = 0)0(x_2 = 0)0(s_{1,1} = 0)0(s_{1,2} = 0)0(s_{2,1} = 0)0(s_{2,2} = 0)$, encodes $\varnothing$. The library strand, $1(x_1 = 1)0(x_2 = 0)1(s_{1,1} = 1)0(s_{1,2} = 0)0(s_{2,1} = 0)0(s_{2,2} = 0)$, encodes the size of the element, 1, and $\{1\}$. The library strand, $0(x_1 = 0)1(x_2 = 1)0(s_{1,1} = 0)0(s_{1,2} = 0)0(s_{2,1} = 0)1(s_{2,2} = 1)$, encodes the size of the element, 2 and $\{2\}$. The library strand, $1(x_1 = 1)1(x_2 = 1)1(s_{1,1} = 1)0(s_{1,2} = 0)0(s_{2,1} = 0)1(s_{2,2} = 1)$, encodes the values of the two elements, 1 and 2, and $\{1, 2\}$. From the execution of Step 3 in Algorithm 1, since it finishes the function of a two-bit parallel adder, DNA sequences encoding the sum of the elements and the carry of the sum are appended onto the tail of the four library strands in tube T_0. Because $\varnothing$ does not contain any element, its sum is zero. Hence, the library strand, $0(x_1 = 0)0(x_2 = 0)0(s_{1,1} = 0)0(s_{1,2} = 0)0(s_{2,1} = 0)0(s_{2,2} = 0)0(y_1 = 0)0(y_2 = 0)0(z_1 = 0)0(y_3 = 0)0(z_2 = 0)0(y_4 = 0)0(z_3 = 0)0(z_4 = 0)0(y_5 = 0)0(z_5 = 0)0(y_6 = 0)0(z_6 = 0)$, encodes

the sum, 0. The subset, $\{1\}$, consists of 1, so its sum is one. Therefore, the library strand, $1(x_1 = 1)0(x_2 = 0)1(s_{1,1} = 1)0(s_{1,2} = 0)0(s_{2,1} = 0)0(s_{2,2} = 0)0(y_1 = 0)0(y_2 = 0)0(z_1 = 0)1(y_3 = 1)0(z_2 = 0)0(y_4 = 0)0(z_3 = 0)0(z_4 = 0)1(y_5 = 1)0(z_5 = 0)0(y_6 = 0)0(z_6 = 0)$, encodes the sum, 1. Since $\{2\}$ includes 2, its sum is two. Hence, the library strand, $0(x_1 = 0)1(x_2 = 1)0(s_{1,1} = 0)0(s_{1,2} = 0)0(s_{2,1} = 0)1(s_{2,2} = 1)0(y_1 = 0)0(y_2 = 0)0(z_1 = 0)0(y_3 = 0)0(z_2 = 0)0(y_4 = 0)0(z_3 = 0)0(z_4 = 0)0(y_5 = 0)0(z_5 = 0)1(y_6 = 1)0(z_6 = 0)$, encodes the sum, 2. The subset $\{1, 2\}$ includes 1 and 2, so its sum is three. Thus, the library strand, $1(x_1 = 1)1(x_2 = 1)1(s_{1,1} = 1)0(s_{1,2} = 0)0(s_{2,1} = 0)1(s_{2,2} = 1)0(y_1 = 0)0(y_2 = 0)0(z_1 = 0)1(y_3 = 1)0(z_2 = 0)0(y_4 = 0)0(z_3 = 0)0(z_4 = 0)1(y_5 = 1)0(z_5 = 0)1(y_6 = 1)0(z_6 = 0)$, encodes the sum, 3. Next, on the execution of Step 4 in Algorithm 1, tube T_b is filled with a library strand, $1(y_5 = 1)1(y_6 = 1)$. From the execution of Step 5 in Algorithm 1, tube T_0 only includes the library strand, $1(x_1 = 1)1(x_2 = 1)1(s_{1,1} = 1)0(s_{1,2} = 0)0(s_{2,1} = 0)1(s_{2,2} = 1)0(y_1 = 0)0(y_2 = 0)0(z_1 = 0)1(y_3 = 1)0(z_2 = 0)0(y_4 = 0)0(z_3 = 0)0(z_4 = 0)1(y_5 = 1)0(z_5 = 0)1(y_6 = 1)0(z_6 = 0)$. Next, from the execution of Step 6 in Algorithm 1, it returns a "yes." Therefore, the answer is $\{1, 2\}$ from the execution of Step 6(a) in Algorithm 1. □

3.7. The Complexity of Algorithm 1

Lemma 7. *Suppose that a finite set S is $\{s_1,\ldots,s_q\}$ and any given positive integer b. The subset-sum problem for S and b can be solved with $O(q \times n)$ biological operations the Adleman–Lipton model from sticker-based solution space, where n is the number of bits for representing the size of every element or b.*

Proof. Algorithm 1 can be applied for solving the subset-sum problem for *S* and *b*. Algorithm 1 includes six main steps. Step 1 takes q *amplify* operations, $2 \times q$ *append* operations, q *merge* operations. Step 2 uses $2 \times n \times q$ *append* operations, q *merge* operations and q *extract* operations. Step 3 takes $(n+q+16 \times n \times q)$ *append* operations, $n \times q$ *merge* operations and $7 \times n \times q$ *extract* operations. Step 4 applies n *append* operations. Step 5 takes $4 \times n$ merge operations, $2 \times n$ extract operations and $5 \times n$ *detect* operations. Step 6 takes one *detect* operation and Step 6(a) takes one *read* operation. Therefore, from the statements above, it is inferred that the time complexity of Algorithm 1 is $O(q \times n)$ biological operations in the Adleman–Lipton model from sticker-based solution space. □

Lemma 8. *Suppose that a finite set S is $\{s_1,\ldots,s_q\}$ and any given positive integer b. The subset-sum problem for S and b can be solved with $O(2^n)$ library strands in the Adleman–Lipton model from sticker-based solution space, where n is the number of bits for representing the size of every element or b.*

Proof. Refer to Lemma 7. □

Lemma 9. *Suppose that a finite set S is $\{s_1,\ldots,s_q\}$ and any given positive integer b. The subset-sum problem for S and b can be solved with $O(c)$ tubes in the Adleman–Lipton model from sticker-based solution space, where c is a constant value.*

Proof. Refer to Lemma 7. □

Lemma 10. *Suppose that a finite set S is $\{s_1,\ldots,s_q\}$ and any given positive integer b. The subset-sum problem for S and b can be solved with the longest library strand, $O(15 \times (q + n + (q + 1) \times n + q \times (n + 1)))$, in the Adleman–Lipton model from sticker-based solution space.*

Proof. Refer to Lemma 7. □

4. Experimental results by simulated DNA computing

From (Roweis et al., 1999; Braich et al., 2003), errors in the separation of the library strands are errors in the computation. This implies that a lower rate of errors of hybridization is needed in the computation. DNA 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 DNA sequences that might encourage unintended probe–library hybridization. To help achieve these goals, the seven constraints for DNA sequences are proposed from (Braich et al., 2003).

Table 1
The number of mutations for the block for satisfying the constraints

The number of mutations	The number of the block	The number of mutations	The number of the block	The number of mutations	The number of the block
1	1	1	2	1	3
1	4	1	5	1	6
2	7	1	8	1	9
1	10	1	11	9	12
1	13	8	14	2	15
1	16	7	17	9	18

From the first constraint, 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 (Mir, 1998). From the second constraint, those long homopolymer tracts may have an unusual secondary structure. The melting temperatures of the probe–library hybrids will be more uniform if none of the probe–library hybrids involve long homopolymer tracts. From the third constraint and the fifth constraint, the probes will bind only weakly where they are not intended to bind. From the fourth constraint and the sixth constraint, the library strands will have a low affinity for themselves. From the seventh constraint, the intended probe–library pairings will have uniform melting temperatures.

We modified the Adleman program (Braich et al., 2003) using a Pentium(R) 4 and 128 MB of main memory. The operating system used is Window 98 and Visual C++ 6.0 compiler. The program modified was applied to generating DNA sequences to solve the subset-sum problem. Because the source code of the two functions *srand*48() and *drand*48() was not found in the *original* Adleman program, we used the standard function *srand*() in Visual C++ 6.0 to substitute function *srand*48() and added the source code for function *drand*48().

The Adleman program was used to construct each 15-base DNA sequence for every bit of the library. For each bit, the program generates two 15-base random sequences ('1' and '0') checking to see if the

Table 2
Sequences chosen were used to represent the 18-bits (blocks)

Bit	5′→3′ DNA sequence	Bit	5′→3′ DNA sequence
x_1^0	***AATTCACAAACAATT***	x_1^1	***TCTCCCTATTTATTT***
x_2^0	***ACTCCTTCCCTACTC***	x_2^1	***CCACCAAACCTAAAC***
$s_{1,1}^0$	***TCTCTCTCTAATCAT***	$s_{1,1}^1$	***CCATCATCTACCTTA***
$s_{1,2}^0$	***ACTCACATACACCAC***	$s_{1,2}^1$	***CAACCTATTAACTTA***
$s_{2,1}^0$	***CTTCTCCACTATACT***	$s_{2,1}^1$	***CCTAAATCTCCAATA***
$s_{2,2}^0$	***AAACTATCATACTTC***	$s_{2,2}^1$	***CTCTCAACAATCAAA***
y_1^0	***TTCAATAAACATTTA***	y_1^1	***CCCTCCTTAACACTT***
z_1^0	***TATTTCTCTCCCAAA***	z_1^1	***CACTATCACTAATAC***
y_2^0	***TATTAACCCAACTAT***	y_2^1	***ACTATAAACCATCCA***
z_2^0	***CCTTTCTAACCTTCA***	z_2^1	***TATCTTTCTTTATCA***
y_3^0	***AACCCAAACTTCTCA***	y_3^1	***ATCCCATATACCTCT***
z_3^0	***ATTACTATCTATAAT***	z_3^1	***TAACCACTCCAACCA***
y_4^0	***AAACTCTACATACAC***	y_4^1	***TTTACCCTCATTACT***
z_4^0	***AATTAACAATCATCT***	z_4^1	***AATTCACTTTCTATC***
y_5^0	***TTACTCTTAACATCT***	y_5^1	***CCACCCTCATCCTAT***
z_5^0	***TTAATCAAATCCCTA***	z_5^1	***CTCTTAATCTCATTC***
y_6^0	***CCTAAATTTCACTAC***	y_6^1	***TTTCTAAACCTCTTC***
z_6^0	***TCCCCACACATTACC***	z_6^1	***ATAAATCCCTTAAAT***

library strands satisfy the seven constraints with the new DNA sequences added (Braich et al., 2003). 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: (A) the constraints are satisfied and the new DNA sequences are then accepted or (B) a threshold for the number of mutations is exceeded and the program has failed and so it exits, printing the sequence found so far. If $(q+n+(q+1)\times n+q\times(n+1))$ bits that satisfy the constraints are found then the program has succeeded and it outputs these sequences.

Making mutations in a new block of the Adleman program picks one of the 15 positions for the mutation at random. This means that many mutations may have nothing to do with the reason that a particular block does not satisfy the chosen constraints, and many mutations may be made without effect. However, making a single base mutation for a new block in the Adleman program may actually be the only path to satisfy the constraints. This is to say that the rate of mutations can be enhanced and DNA sequences satisfying the constraints can be found in a reasonable amount of time. Therefore, from the Adleman program, a lower rate of mutations and a lower rate of errors of hybridization can be simultaneously achieved.

Consider the finite set S and any given positive integer b in Eq. (1). The finite set S is $\{1, 2\}$. The value for b is three. The number of mutations for DNA sequences, satisfying the constraints, is shown in Table 1. DNA sequences generated by the modified Adleman program are shown in Table 2. With the nearest neighbor parameters, the Adleman program was used to calculate the enthalpy, entropy, and free energy for the binding of each probe to its corresponding region on a library strand. The energy used was shown in Table 3. 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 25 kcal/mol and the delta G for the probe binding a '0' is estimated to be 24.3 kcal/mol. The program also figured out the average and standard deviation for the enthalpy, entropy and free energy over all probe/library strand interactions. The energy levels are shown in Table 4.

The Adleman program was employed for computing the distribution of the different types of potential mishybridizations. The distribution of the types of potential mishybridizations is the absolute frequency of a

Table 3
The energy for binding of each probe to its corresponding region on a library strand

Bit	Enthalpy energy (H)	Entropy energy (S)	Free energy (G)
x_1^1	114.4	299.4	25
x_1^0	107.8	278.6	24.3
x_2^1	114.3	291.2	27.1
x_2^0	109.1	279	25.9
$s_{1,1}^1$	105.2	270.5	24.4
$s_{1,1}^0$	97.3	252.3	22.1
$s_{1,2}^1$	108.4	286.3	22.6
$s_{1,2}^0$	94.7	239.8	22.8
$s_{2,1}^1$	111.1	288.3	25
$s_{2,1}^0$	101.9	266	22.4
$s_{2,2}^1$	101.3	258	24.1
$s_{2,2}^0$	102.1	269.7	21.4
y_1^1	114.1	292.1	26.8
y_1^0	107.5	282.2	22.9
z_1^1	96.6	255.3	20.1
z_1^0	111.2	285	26.1
y_2^1	107.4	277.5	24.4
y_2^0	111.1	292	23.7
z_2^1	103.8	272.6	22.3
z_2^0	111.3	285.7	25.9
y_3^1	109.9	284.1	25.2
y_3^0	109.8	278.2	26.6
z_3^1	107.5	269.6	26.8
z_3^0	104.3	283.6	19.5
y_4^1	109.9	285.5	24.5
y_4^0	97.6	255.8	20.9
z_4^1	104.3	273	22.7
z_4^0	105.3	275.5	22.9
y_5^1	112.1	282.8	27.8
y_5^0	101.3	266.6	21.5
z_5^1	102.1	266	22.7
z_5^0	112.4	291.5	25.3
y_6^1	109.2	283.7	24.4
y_6^0	108	282.9	23.3
z_6^1	118.7	313	25.2
z_6^0	110.2	275.3	27.9

Table 4
The energy over all probe/library strand interactions

	Enthalpy energy (H)	Entropy energy (S)	Free energy (G)
Average	107.033	277.461	24.0694
Standard deviation	5.47827	14.0834	2.09149

probe-strand match of length k from 0 to the bit length 15 (for DNA sequences) where probes are not supposed to match the strands. The distribution was, subsequently, 628, 1492, 3124, 5517, 7719, 8125, 6623, 4264, 2115, 842, 253, 54, 1, 0, 0 and 0. It is pointed out from the last three zeros that there are 0 occurrences where a probe matches a strand at 13, 14 or 15 places. Hence, the number of matches peaks at 5 (8125). That is to say that there are 8125 occurrences where a probe matches a strand at 5 places.

5. Conclusions and future research work

The proposed algorithm (Algorithm 1) for solving the subset-sum problem is based on biological operations in the Adleman–Lipton model and the solution space of stickers in the sticker-based model. This modified algorithm has several advantages from the Adleman–Lipton model and the sticker-based model. First, the proposed algorithm actually has a lower rate of errors for hybridization after we modified the Adleman program to generate good DNA sequences for constructing the solution space of stickers to the subset-sum problem. The basic biological operations in the Adleman–Lipton model were employed to finish the function of an n-bit parallel adder and the function of an n-bit parallel comparator for solving the subset-sum problem. Secondly, the basic biological operations in the Adleman–Lipton model had been performed in a fully automated manner in their lab. The full automation manner is essential not only for the speedup of computation but also for error-free computation. Thirdly, in Algorithm 1 for solving the subset-sum problem, the number of tubes, the longest length of DNA library strands, the number of DNA library strands and the number of biological operations, respectively, are $O(c)$, $O(15 \times (q + n + (q + 1) \times n + q \times (n + 1)))$, $O(2^n)$ and $O(q \times n)$. This implies that the proposed algorithm can be easily performed in a fully automated manner in a lab. Furthermore, this presented algorithm generates 2^n library strands, which satisfy the constraints in (Braich et al., 2003) and correspond to 2^n possible solutions. This allows the proposed algorithm to be applied to a larger instance of the subset-sum problem. Fourthly, since an n-bit parallel adder can be implemented in a fully automated manner in a lab, this seems to imply that mathematical operations can be performed on a DNA-based computing.

Any multiplication operation can be accomplished through many addition operations. We proposed an n-bit parallel adder to implement the *addition* operation. This indicates that it seems to be reasonable for extending an n-bit parallel adder to finish multiplication operation. Simultaneously, this implies that NP-complete problems with multiplication operations can be solved. In the future, our primary work is to solve other outstanding NP-complete problems with multiplication operations that were irresolvable using the Adleman–Lipton model and the sticker-based model.

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 11th 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., Leonard, M., 2003. 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.

Chang, W.-L., Guo, M., 2002a. Solving the dominating-set problem in Adleman–Lipton's model. In: The Third International Conference on Parallel and Distributed Computing, Applications and Technologies, Japan, pp. 167–172.

Chang, W.-L., Guo, M., 2002b. Solving the clique problem and the vertex cover problem in Adleman–Lipton's model. In: IASTED International Conference, Networks, Parallel and Distributed Processing, and Applications, Japan, pp. 431–436.

Chang, W.-L., Guo, M., 2002c. Solving NP-complete problem in the Adleman–Lipton model. In: The Proceedings of 2002 International Conference on Computer and Information Technology, Japan, pp. 157–162.

Chang, W.-L., Guo, M., 2002d. Resolving the 3-dimensional matching problem and the set-packing problem in Adleman–Lipton's model. In: IASTED International Conference, Networks, Parallel and Distributed Processing, and Applications, Japan, pp. 455–460.

Chang, W.-L., Guo, M., 2003. Solving the Set-cover Problem and the Problem of Exact Cover by 3-sets in the Adleman–Lipton model. BioSystems 72, 263–275.

Chang, W.-L., Guo, M., Ho, M., 2003. Solving the Set-splitting Problem in Sticker-based Model and the Adleman–Lipton Model. In: The 2003 International Symposium on Parallel and Distributed Processing and Applications, LNCS 2745, Aizu City, Japan, July 2003.

Cook, S.A., 1971. The complexity of theorem-proving procedures. In: Proceedings of Third Annual ACM Symposium on Theory of Computing. ACM, New York, pp. 151–158.

Cormen, T.H., Leiserson, C.E., Rivest, R.L., 2003. Introduction to Algorithms.

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

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

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

Karp, R.M., 1972. Reducibility among combinatorial problems. In: Complexity of Computer Computation. Plenum Press, New York, pp. 85–103.

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

LaBean, M.C., Reif, T.H., Seeman, J.H., 2003. 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.

Mir, K., 1998. A restricted genetic alphabet for DNA computing. In: Baum, E.B., Landweber, L.F. (Eds.), DNA Based Computers II: DIMACS Workshop, June 10–12, 1996, DIMACS: Series in Discrete Mathematics and Theoretical Computer Science, vol. 44, Providence, RI, pp. 243–246.

Morimoto, N., Arita, M., Suyama, A., 1999. Solid phase DNA solution to the Hamiltonian path problem. DIMACS (Ser. Discrete Math. Ther. Comput. Sci.) 48, 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-Verlag, New York, ISBN: 3-540-64196-3.

Perez-Jimenez, M.J., Sancho-Caparrini, F., 2001. Solving knapsack problems in a sticker based model. In: 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 (meeting at Leiden, Holland, June).

Roweis, S., Winfree, E., Burgoyne, R., Chelyapov, N.V., Goodman, M.F., Rothemund, P.W.K., Adleman, L.M., 1999. A sticker based model for dna computation. In: Landweber, L., Baum, E. (Eds.), Second Annual Workshop on DNA Computing, Princeton University. DIMACS: Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society, pp. 1–29.

Shin, S.-Y., Zhang, B.-T., Jun, S.-S., 1999. Solving traveling salesman problems using molecular programming. Proc. Cong. Evol. Comput. (CEC99) 2, 994–1000.

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