J Math Chem (2008) 44:172–183
DOI 10.1007/s10910-007-9299-6

ORIGINAL PAPER

Molecular solution to the optimal linear arrangement problem based on DNA computation

Xingchang Liu · Xiaofan Yang · Yuan Yan Tang

Received: 3 March 2007 / Accepted: 12 July 2007 / Published online: 21 September 2007

Abstract Due to massive parallelism, enormous memory storage and very low energy consumption, biomolecular operations have been suggested to solve various NP-hard problems that are beyond the capability of the fastest known digital computer. The optimal linear arrangement (OLA) problem is a well-known NP-hard combinatorial optimization problem. Based on a DNA computational model, this paper describes a novel algorithm for the OLA problem, which is executed in $O(n^3 \log_2 n)$ DNA operations on tubes of $(nK + n + m + L + 1)$-bits DNA strands, where $K = \lceil \log_2 n \rceil$ and $L = \lceil \log_2 (nm) \rceil + 1$. With the advance in molecular biology techniques, this algorithm may be of practical utility.

Keywords DNA computation · DNA algorithm · Adleman-Lipton-sticker model · Optimal linear arrangement problem · NP-hardness

1 Introduction

DNA computation uses DNA as information storage and biochemical operations to process the information. It can be used to solve computationally intractable (technically, NP-complete or NP-hard) problems due to its advantages over conventional digital computing: massive parallelism, enormous memory storage and low energy consumption. Since the seminal work by Adleman [1] that describes how to solve a seven-node instance of a well-known NP-hard problem (the directed Hamiltonian

X. Liu · X. Yang (✉) · Y. Y. Tang
College of Computer Science, Chongqing University, Chongqing, 400044, China
e-mail: xf_yang1964@yahoo.com

X. Liu
Department of Logistical, Information Engineering, Logistical Engineering University, Chongqing, 400016, China

path problem) via biological operations, DNA computation has received considerable interests from researchers. In particular, several typical DNA computational models have already been established (say, the *Adleman-Lipton* model [1,2], the *restriction-enzyme* model [3], the *sticker* model [4], the *surface-based* model [5], the *self-assembly* model [6] and the *hairpin* model [7]). Based on these models, a number of NP-complete (or NP-hard) problems have been solved at least theoretically [1–3,8–14]. In order to fully understanding the power of biological computation, it is worthwhile to try to solve more kinds of computationally intractable problems with the aid of DNA operations.

The *optimal linear arrangement* (OLA) problem, which was introduced by Harper to design error-correcting codes with minimal average absolute errors on certain classes of graph, is to find a permutation π of vertices V of a given graph such that the cost $\sum_{(i,j)\in E} |\pi(i) - \pi(j)|$ is minimum [15,16]. Some graph layout problems, such as bandwidth and cutwidth, are the variations of the OLA. The primary applications of the OLA are in the areas of circuit design and circuit layout [17–20]. Additionally, The OLA also plays an important role in computational biology [21], graph drawing [22], and so on. The OLA is well known to be NP-hard, and in some sense even more difficult than NP-hardness alone would indicate [23,24]. As a result, various heuristic algorithms have been devised for the OLA [17,25–27].

Motivated by the above mentioned work, this paper presents a molecular algorithm for the OLA problem based on a combination of Adleman-Lipton model and the sticker model. The proposed algorithm suggests a promising solution to the OLA for it requires only $O(n^3 \log_2 n)$ DNA operations on tube of $(nK + n + m + L + 1)$-bits DNA strands, where $n = |V|$, $m = |E|$, $K = \lceil \log_2 n \rceil$, and $L = \lceil \log_2 (nm) \rceil + 1$. It may play an important role in practice, when further advances in biological techniques lead to an efficient implementation of DNA computer.

This paper is organized as follows. Section 2 formally describes the *Adleman-Lipton-sticker* model and the OLA problem. Sections 3 and 4 propose a novel DNA algorithm for the OLA problem and analyze its complexity. Section 5 closes this work by some summary remarks.

2 Preliminary knowledge

In this paper, we formally describe the *Adleman-Lipton-sticker* model of DNA computation. We then formalize the *OLA* problem.

2.1 DNA computation

A DNA molecule is a polymer constructed from monomers called deoxyribonucleotides, which are strung together in a strand like beads on a necklace. A schematic representation of a DNA molecule is shown in Fig. 1. Every deoxyribonucleotide is comprised of three parts: a ribose group, a phosphate group and a nitrogenous base. The ribose has five carbon atoms, which are numbered from $1'$ to $5'$. Within the ribose there is a hydroxyl group attached to the $3'$ carbon. The base is attached to the $1'$ carbon, and the phosphate group is attached to the $5'$ carbon. In DNA molecules, nucleotides are only distinguished from their bases, which are adenine, guanine, cytosine, and

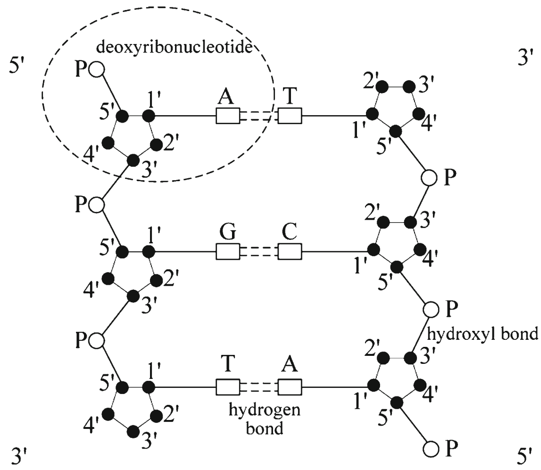

Fig. 1 A schematic representation of a DNA molecule

thymine, respectively abbreviated A, G, C, and T. Therefore, nucleotides are simply represented as A, G, C, or T nucleotides, according to their corresponding bases.

Any single strand of DNA is linked by a backbone that is formed by the alternating phosphate and ribose of each nucleotide, in which the $5'$-phosphate group of one nucleotide is joined with the $3'$-hydroxyl group of the other. This gives the DNA molecule a direction from $5'$-phosphate group (denoted by $5'$ end) to $3'$-hydroxyl group (denoted by $3'$ end), or reverse direction from $3'$ end to $5'$ end.

DNA is best known for double-helix bonding. Nucleotides in respective DNA strands are attracted each other by hydrogen bond, which is formed by the base of one nucleotide interacting with the base of the other. This attraction exists only in restrict pairs of bases: A matches T, and C matches G. This pairing principle is called the *Watson–Crick* complement rule. Two strands of DNA can form the most stable double helix, only if the respective bases are the Watson–Crick complements of each other, and also $3'$ end matches $5'$ end.

Along with the development of molecular biology techniques, the DNA strands can be quickly and cheaply synthesized. Thus, they can store information in the form of four-letter strings (A, G, C and T) at molecular level. Founded on above idea, DNA computation proceeds in three phases: *first*, generate a data pool of DNA strands that encode all possible solutions to the studied problem; *second*, by employing molecular biology laboratory techniques, orderly apply a series of DNA operations on DNA strands to in large exclude the DNA strands that do not satisfy logic constraints of the problem; *third*, detect whether a result set contains at the least one DNA strand, if do, describe it, *i.e.*, readout answer. The second step is a typical data-parallel computation that greatly accelerates a tedious computing process for a hard problem.

2.2 The Adleman-Lipton-sticker model

The *Adleman-Lipton-sticker* model is a DNA computational model, which fully utilizes the advantages of both the Adleman-Lipton model and the sticker model. Below is a detailed description of this hybrid model.

2.2.1 Representation of information

Under the *Adleman-Lipton-sticker* model, information (a bit string) is represented by the partial duplex DNA strand called memory complex. Memory complex involves two basic groups of single stranded DNA molecules: the longer memory strands and the shorter sticker strands (or simply *sticker*). A typical memory strand is divided into N non-overlapping regions (known as *bits*) with B bases each (typically, $B := 20$). Corresponding to N-bits memory strand, N different sticker is needed. Each sticker is B bases long and is complementary to one and only one of the N bits according to the Watson–Crick complement rule. A bit of a given memory strand assumes value 1 or 0 depending on whether its corresponding sticker is annealed to this region or not. In this way, a bit string of $\{0, 1\}^N$ is represented by a N-bits memory strand with stickers annealed only at required regions. Accordingly, a large set of bit strings is represented by a collection of memory complexes, called a tube.

According to Sect. 2.1, the first step of DNA computation is to generate solution space of DNA strands. Consider a problem with M-bits input. It is clear that there are 2^M possible solutions and, hence, the memory strand encoding the possible solutions must have more than M bits. The first Mbits, we called *solution space*, represent the encoding of the possible solution and are "randomly" annealed with corresponding stickers at the first step. The remaining $N - M$ bits, we called *working space*, are used for intermediate storage and initially zero.

Although errors are inevitable in biochemical operations, error rates can be reduced to tolerable levels. Under the Adleman-Lipton-sticker model, the error-resistant ability heavily depends on the DNA sequence of the memory strand. Up to the present, the problem of strand design has been an open question. For more details, refer to [4,28–30].

Design of the memory strand may be difficult, whereas, once an N-bits strand found, it can be used and reused for any problem requiring N or fewer bits. In a sense, the design is simplified for functionality of the strand can be designed and tested once and for all.

2.2.2 DNA operations

Under the *Adleman-Lipton-sticker* model, several DNA operations on tubes are defined to implement special algorithms, just as mathematical operations on a multiset of bit strings. The following are some operations used in this paper.

There are five principal operations during computing: *merge, extract, set, clear, and discard*.

- $T := \text{merge}(T_1, T_2)$. Given two tubes T_1 and T_2, get a tube T containing all strands in T_1 or T_2.
- $(T_0, T_1) := \text{extract}(T, i)$. Given a tube T and a bit index i, get two tubes T_0 and T_1, where T_0 and T_1 contains all those strands in T with bit i set as 0 and 1, respectively.
- $T := \text{set}(T_0, i)$. Given a tube T_0 and a bit index i, get a tube T by setting bit i of all strands in T_0 as 1.

- T := clear(T_0, i). Given a tube T_0 and a bit index i, get a tube T by setting bit i of all strands in T_0 as 0.
- discard(T). Given a tube T, discard all strands in T.

During the preparation of the initial tube, three additional operations are necessary: *make*, *amplify,* and *separate*.

- T := make(co). Given the code co of a DNA sequence, get a tube T of exactly one single-stranded DNA molecule that is encoded with co.
- T := amplify(T_0, p). Given a tube T_0 and a positive integer p, get a tube T that contains p copies of every DNA strand in T_0.
- (T_1, T_0) := separate(T). Given a tube T, get two test tubes T_1 and T_0, each of which contains one half of the contents in T.

For the purpose of obtaining the final result, we need two different operations: *detect* and *read*.

- *bool* := detect(T): Given a tube T, get a Boolean variable *bool*, which assumes value *yes* or *no* according as there exists a DNA strand in T or not.
- s := read(T): Given a tube T of at least one DNA strand, get a strand s in T.

For the implementation details of these biochemical operations, refer to [4].

2.3 The OLA problem

Consider a undirected simple graph $G = (V, E)$, where $V = \{v_1, v_2, \ldots, v_n\}$ and $E = \{e_1, e_2, \ldots, e_m\}$. Let $n = |V|$, $m = |E|$. The incidence matrix of G is an $n \times m$ 0–1 matrix $M(G) = [m_{ij}]$, where $m_{ij} = 1$ or 0 according as the vertex v_i is incident to the edge e_j or not.

A *linear arrangement* of the vertices, V, is a permutation $\pi = \pi(1)\,\pi(2)\ldots\pi(n)$ of $\{1, 2, \ldots, n\}$. The vertex v_i is assigned the $\pi(i)$th position in the linear arrangement. The cost of a permutation π is:

$$\text{cost}(\pi) = \sum_{(i,j)\ni E} |\pi(i) - \pi(j)| \tag{1}$$

The *optimal linear arrangement* (OLA) problem is to obtain a π with minimum cost.

3 A DNA algorithm for the OLA problem

3.1 Basic idea

The basic idea behind our DNA algorithm for the OLA problem is to find an optimal arrangement by checking all n-permutations of $\{1, 2, \ldots, 2^{\lceil \log_2 n \rceil}\}$ with repetition allowed (namely, $P^R(2^{\lceil \log_2 n \rceil}, n)$) by brute force. Specifically, the proposed algorithm consists of four steps:

Step 1: Construct set T of $P^R(2^{\lceil \log_2 n \rceil}, n)$ permutations;

Step 2: Scan all elements of every permutation in T orderly, exclude illegal ones with elements reduplicated or out of n, thus, get all linear arrangements of vertices V;

Step 3: Accumulate contribution of element to the cost of the arrangement while scanning in *step 2*;

Step 4: Pick out an arrangement in T so that the cost is minimum.

3.2 Strand design

To implement above idea, our DNA algorithm will perform operations on tubes of $(nK+n+m+L+1)$-bits DNA strands, where $K = \lceil \log_2 n \rceil$, $L = \lceil \log_2(nm) \rceil + 1$. $f(x) = \lceil \log_2 x \rceil$ is the number of bits of a nonnegative binary integer x. Furthermore, every such strand is partitioned into *solution space* and *working space* that is composed of *vertex space*, *edge space*, and *cost space* (Fig. 2):

- The *solution space*, which is composed of the 1st through (nK)th bits. This space is subdivided into n segments with K bits each. For $1 \leq j \leq n$, The jth segment set to i (obviously, $0 \leq i \leq 2^K - 1$) represents that v_{i+1} is assigned the jth position, *i.e.*, $\pi(i+1) = j$. Therefore, all possible $P^R(2^K, n) = 2^{nK}$ arrangement candidates are recorded.
- The *vertex space*, which is composed of the (nK + 1)th through the $(nK + n)$th bits. For $1 \leq i \leq n$, the ith bit in this space will be logically viewed as VS(i). Bit VS(i) set to 1 represents that v_i is already in the permutation. In $P^R(2^K, n)$ permutations, all those unfeasible linear arrangements of graph G include two situations: elements are repetitive or out of n. This space can be used to eliminate the former.
- The *edge space*, which is composed of the $(nK + n + 1)$th through the $(nK + n + m)$th bits. For $1 \leq k \leq m$, the kth bit in this space will be logically viewed as ES(k). According to equation (1), if $e_k = (i, j)$ and $\pi(i) < \pi(j)$, the contribution of v_i to the cost of the arrangement is $-\pi(i)$ and v_j is $+\pi(j)$. Bit ES(k) set to 1 represents that the $\pi(i)$th segment of the solution space has been checked while scanning the permutation. Due to the design of the solution space, the $\pi(i)$th segment is always checked before the $\pi(j)$th.
- The *cost space*, which is composed of the $(nK + n + m + 1)$th through the $(nK + n + m + L + 1)$th bits. For $1 \leq i \leq L + 1$, the ith bit in this space will be logically viewed as CS(i). Bits CS(1) $\sim$ CS(L) will be used to calculate and store the value of the cost in form of complementary offset binary, where sign is

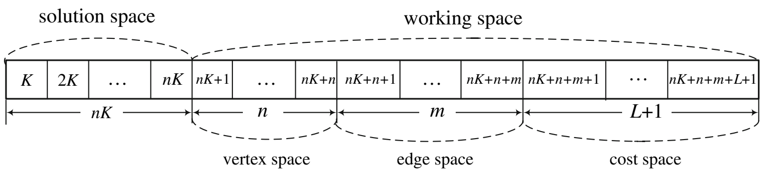

Fig. 2 An $(nK, n+m+L+1)$ DNA strand

assigned to CS(1). Bit CS($L+1$) will be used to store related carry information. In spite of the fact that the upper bound on OLA for any simple graph G of order n is $\frac{(n-1)n(n+1)}{6}$ [25], L bits are sufficient to store intermediate data of cost.

For our purpose, the code of the $(nK+n+m+L+1)$-bits memory strand is assumed to be known. Henceforth, such a DNA strand is called as an $(nK, n+m+L+1)$ strand. Given an $(nK, n+m+L+1)$ strand s, we let $s(p, q)$ denote the binary string corresponding to the pth through qth bit regions of s, and s(CS(1) : CS(L)) the integer formed by the values stored in bits CS(1) through CS(L).

To implement our DNA algorithm, several DNA subroutines are developed in subsequent four subsections. Their time complexities are analyzed in terms of the numbers of DNA operations involved.

3.3 DNA initialization

Subroutine $T :=$ **dna_init**(n, m, r)
Input: n, m, r: three positive integers.
Output: T: a tube that contains r copies of each $(nK, n+m+L+1)$ strand that has 1 or 0 in each bit of the solution space and has 0 in each bit of the working space.

```
   begin
1. T := make(code(nK + n + m + L + 1));
2. T := amplify(T, r × 2^{nK});
3. for i := 1 to nK
4.   (T0, T1) := separate(T);
5.   T1 := set(T1, i);
6.   T := merge(T1, T0);
7. end {for i};
   end.
```

Remark The parameter r is used to regulate the error-tolerance of our algorithm. The error-resistant ability of the proposed algorithm can be further improved by employing the techniques proposed by Boneh and Lipton [31] and Karp et al. [32].

By inspection of this subroutine, we get

Lemma 1 *Subroutine dna_init is executed in* $3nK+2$ *tube operations.*

3.4 DNA division

Subroutine $(T_1, T_2, \ldots, T_n) :=$ **dna_divi**(T_0, k)
Input: T_0: a test tube of $(nK, n+m+L+1)$ DNA strands;
k: an integer that satisfies $1 \leq k \leq n$.
Output: $(T_1, T_2, \ldots, T_n)$: n tubes, where T_i only contains all those stands in T_0 with the kth segment of solution space set as $(i-1)$.

begin
$R := 0$;
rename T_0 as T_1;
for $i := 0$ to $K - 1$
  for $j := 0$ to R step 2^{K-i}
    $(T_{j+1}, T_{j+2^{K-i-1}+1}) :=$ extract$(T_{j+1}, K(k-1) + i + 1)$;
  end {for j};
  if $n > (R + 2^{K-i-1})$, then $R := R + 2^{K-i-1}$;
end {for i};
if $(j + 2^{K-i-1} + 1) > n$, then discard$(T_{j+2^{K-i-1}+1})$;
end.

The executing of subroutine *dna_divi* can be intuitively described in form of a binary tree, as shown in Fig. 3 (for $n = 11$). By relative theorem about binary tree and inspection of this subroutine, we derive

Lemma 2 *Subroutine dna_divi is executed in* $n + 2$ *tube operations.*

3.5 DNA addition

Given an integer d, we let $[d]_c$ denote its complementary offset binary representation.

Subroutine $T :=$ **dna_add**(T_0, d)
Input: T_0: a tube of $(nK, n + m + L + 1)$ DNA strands;
d: an integer;
Output: T: a tube obtained by modifying every strand s in T_0 in this way:
$s(\mathrm{CS}(1) : \mathrm{CS}(L)) := s(\mathrm{CS}(1) : \mathrm{CS}(L)) + [d]_c$

begin
$T :=$ clear$(T_0, \mathrm{CS}(L + 1))$;
for $i := 0$ to $L - 1$
  $(T_0, T_1) :=$ extract$(T, \mathrm{CS}(L - i))$;
  $(T_{10}, T_{11}) :=$ extract$(T_1, \mathrm{CS}(L + 1))$;
  $(T_{00}, T_{01}) :=$ extract$(T_0, \mathrm{CS}(L + 1))$;

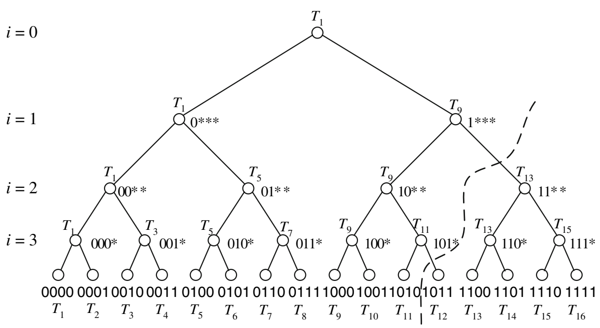

Fig. 3 An intuitive describe of the executing of subroutine dna_divi (for $n = 11$)

if the $(i+1)$th least significant bit of $[d]_c$ is 1, then
$T_{10} := \text{clear}(T_{10}, \text{CS}(L-i))$;
$T_{10} := \text{set}(T_{10}, \text{CS}(L+1))$;
$T_{00} := \text{set}(T_{00}, \text{CS}(L-i))$;
end {if};
if the $(i+1)$th least significant bit of $[d]_c$ is 0, then
$T_{11} := \text{clear}(T_{11}, \text{CS}(L-i))$;
$T_{01} := \text{set}(T_{01}, \text{CS}(L-i))$;
$T_{01} := \text{clear}(T_{01}, \text{CS}(L+1))$;
end {if};
$T_1 := \text{merge}(T_{11}, T_{10})$;
$T_0 := \text{merge}(T_{01}, T_{00})$;
$T := \text{merge}(T_1, T_0)$;
end {for i};
end.

Lemma 3 *Subroutine dna_add is executed in* $9L+1$ *tube operations.*

3.6 DNA optimization

Subroutine $s :=$ **dna_opt**(T)
Input: T: a nonempty test tube of $(nK, n+m+L+1)$ DNA strands.
Output: s: a DNA strand in T such that $s(\text{CS}(2):\text{CS}(L))$ attains the minimum.

begin
for $i = 2$ to L
$(T_0, T_1) := \text{extract}(T, \text{CS}(i))$;
if $\text{detect}(T_0) = \text{yes}$, then rename T_0 as T;
else rename T_1 as T;
end {for i};
$s := \text{read}(T)$;
end.

Because of positive cost and definition of complementary offset binary, we conclude that a strand with minimum cost can be found after subroutine *dna_opt*.

Lemma 4 *Subroutine dna_opt is executed in* $2L-1$ *tube operations.*

3.7 Complete description of the DNA algorithm

Based on the previous discussions, we are in a position to describe a DNA algorithm for the OLA problem.

Algorithm ***BS*** := **dna_ola**(G, r)
Input: G: an undirected simple graph, with n vertices, m edges and incident matrix M.
r: a nonnegative integer, which is used as the error-resistant control parameter.

Output: a binary string $BS = b_1b_2, \ldots, b_{nK}$ that represents an optimal linear arrangement of given graph G.

```
begin
K := ⌈log2 n⌉;
L := ⌈log2 (nm)⌉ + 1;
T1 := dna_init(n, m, r);
for k := 1 to n
  (T1, T2, ..., Tn) :=dna_divi(T1, k);
  for i := 1 to n
    (Ti0, Ti1) := extract(Ti, VS(i));
    discard(Ti1);
    Ti0 := set(Ti0, VS(i));
    rename Ti0 as Ti;
    for j := 1 to m
      if mij = 1, then
        (Ti0, Ti1) := extract(Ti, ES(j));
        Ti0 := dna_add(Ti0, −k);
        Ti0 := set(Ti0, ES(j));
        Ti1 := dna_add(Ti1, +k);
        Ti := merge(Ti0, Ti1);
      end {if};
    end {for j};
  end {for i};
  for i := 2 to n, T1 := merge(T1, Ti);
end {for k};
s := dna_opt(T1);
return(s(1, nK));
end.
```

4 The complexity of the proposed DNA algorithm

There are several criterions to measure a DNA algorithm:

- The solution space size, which determining the volume of tube. We call it *volume complexity*;
- The maximal length of the DNA molecular. In Adleman-Lipton-sticker model, measured as the length of the memory strand. We call it *molecular complexity*;
- The numbers of tubes used during the algorithm is executed. We call it *space complexity*;
- The numbers of operations performed during the algorithm is executed. We call it *time complexity*.

The following two theorems, which follow directly from Sect. 3.2, characterize the volume complexity and the molecular complexity of the algorithm *dna_ola*, respectively.

Theorem 1 *For a graph G with n nodes and m edges, algorithm dna_ola has a solution space of size* 2^{nK}.

Theorem 2 *For a graph G with n nodes and m edges, the memory strand used in algorithm dna_ola consists of* $nK + n + m + L + 1$ *bit regions.*

Theorem 3 and Theorem 4 describe the space complexity and the time complexity of the algorithm *dna_ola*, respectively.

Theorem 3 *For a graph G with n nodes and m edges, algorithm dna_ola requires 2n tubes.*

Proof After statement 5, tube T_1 is divided into n tubes. For each tube T_i, two tubes are required to accomplish statements 7–17. □

Theorem 4 *For a graph G with n nodes and m edges, algorithm dna_ola is executed in* $18nmL + 2n^2 + 5nm + 3nK + 4n + 2L + 1$ *tube operations.*

Proof By Lemma 1, statement 3 requires $3nK + 2$ tube operations.

By Lemma 2, statement 4 costs $n + 2$ tube operations.

Since statements 7–19 can be executed simultaneously in n tubes, the loop represented by statements 6–20 is carried out in $18mL + 5m + 3$ tube operations by observation and in review of Lemma 3.

Therefore, the execution of the loop represented by statements 4–22 needs $18nmL + 2n^2 + 5nm + 4n$ tube operations.

By Lemma 4, statement 23 is executed in $2L - 1$ tube operations.

The claimed result follows. □

Note that $m \leq \frac{n(n-1)}{2}$, $K = \lceil \log_2 n \rceil$, and $L = \lceil \log_2 (nm) \rceil + 1$, thus, the algorithm dna_ola can be executed in $O(n^3 \log_2 n)$ tube operations.

5 Summary

Under the Adleman-Lipton-sticker model, a DNA algorithm for the OLA problem has been presented. This algorithm is theoretically effective for it is executed in $O(n^3 \log_2 n)$ tube operations on tubes of $(nK + n + m + L + 1)$-bits DNA strands. With further advance in molecular biology techniques, it may play an important role in practice. Moreover, the work of this paper provides more evidence for the ability of DNA computation to solve the NP-hard problems.

Acknowledgments This work is financially supported by New Century Excellent Talent of Educational Ministry of China (Grant No. NCET-05–0759), Doctorate Funds of Educational Ministry of China (Grant No. 20050611001) and Natural Science Funds of Chongqing CSTC (Grant Nos. 2006BB2331, 2005BB2191).

References

1. L.M. Adleman, Molecular computation of solution to combinatorial problems. Science **266**, 1021–1024 (1994)
2. R.J. Lipton, DNA solution of hard computational problems. Science **268**, 542–545 (1995)
3. Q. Ouyang, P.D. Kaplan, S. Liu, A. Libchaber, DNA solution of the maximal clique problem. Science **278**, 446–449 (1997)

4. 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. J. Comput. Biol. **5**, 615–629 (1998)
5. L.M. Smith, R.M. Corn, A.E. Condon, M.G. Lagally, A.G. Frutos, Q. Liu, A.J. Thiel, A surface-based approach to DNA computation. J. Comput. Biol. **5**, 255–267 (1998)
6. E. Winfree, F. Liu, L.A. Wenzler, N.C. Seeman, Design and self-assembly of two dimensional DNA crystals. Nature **394**, 539–544 (1998)
7. K. Sakamoto, H. Gouzu, K. Komiya, D. Kiga, S. Yokoyama, T. Yokomori, M. Hagiya, Molecular computation by DNA hairpin formation. Science **288**, 1223–1226 (2000)
8. E. Bach, A. Condon, E. Glaser, C. Tanguay, DNA models and algorithms for NP-complete problems. J. Comp. Syst. Sci. **57**, 172–186 (1998)
9. Q. Liu, L. Wang, A.G. Frutos, A.E. Condon, R.M. Corn, L.M. Smith, DNA computing on surfaces. Nature **403**, 175–179 (2000)
10. S.-Y. Shin, B.-T. Zhang, S.-S. Jun, Solving traveling salesman problems using molecular programming. in *Proceeding of Congress on Evolutionary Computation* (1999) 994–1000
11. D. Xiao, W. Li, Z. Zhang, L. He, Solving maximum cut problems in the Adleman-Lipton model. BioSystems **82**, 203–207 (2005)
12. C.-N. Yang, C.-B. Yang, A DNA solution of SAT problem by a modified sticker model. BioSystems **81**, 1–9 (2005)
13. C.-W. Yeh, C.-P. Chu, K.-R. Wu, Molecular solutions to the binary integer programming problem based on DNA computation. BioSystems **83**, 56–66 (2006)
14. K.-H. Zimmermann, Efficient DNA sticker algorithms for NP-complete graph problems. Comp. Phys. Commun. **144**, 297–309 (2002)
15. D. Adolphson, T.C. Hu, Optimal linear ordering. SIAM J. Appl. Math. **25**(3), 403–423 (1973)
16. L.H. Harper, Optimal assignments of numbers to vertices. J. Soc. Indust. Appl. Math. **12**(1), 131–135 (1964)
17. J. Bhasker, S. Sahni, Optimal linear arrangement of circuit components. J. VLSI Comp. Syst. **2**(1–2), 87–109 (1987)
18. C.K. Cheng, Linear placement algorithms and applications to VLSI design. Networks **17**(4), 439–464 (1987)
19. J. Díaz, J. Petit, M. Serna, A survey of graph layout problems. ACM Comput Surveys **34**(3), 313–356 (2002)
20. W.-L. Lin, M. Sarrafzadeh, A linear arrangement problem with applications. in *1995 IEEE Int Symp Circuits Syst(ISCAS'95)* **1**, 57–60 (1995)
21. R.M. Karp, Mapping the genome: some combinatorial problems arising in molecular biology. in *Proceedings of the 25th Annual ACM Symposium on Theory of Computing* 278–285 (1993)
22. F. Shahrokhi, O. Sỳkora, L.A. Śzekly, I. Vrto, On bipartite drawings and the linear arrangement problem. SIAM J Comp **30**(6), 1773–1789 (2001)
23. M.R. Garey, D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness* (Freeman, San Francisco, California, 1979)
24. S. Even, Y. Shiloach, NP-completeness of several arrangement problems. *Technical Report #43* (Computer Science Department, Technion, Haifa, Israel, 1975)
25. S. Horton, The optimal linear arrangement problem: algorithms and approximation. (*PhD Thesis*, Georgia Institute of Technology, 1997)
26. A.J. McAllister, A new heuristic algorithm for the linear arrangement problem. *Technical Report 99 126a* (Faculty of Computer Science, University of New Brunswick, 1999)
27. T. Poranen, A genetic hillclimbing algorithm for the optimal linear arrangement problem. Foudamenta Informaticate **68**(4), 333–356 (2005)
28. A. Brenneman, A. Condon, Strand design for biomolecular computation. Theor. Comp. Sci. **287**, 39–58 (2002)
29. P. Gaborit, O.D. King, Linear constructions for DNA codes. Theor. Comp. Sci. **334**(1–3), 99–113 (2005)
30. A. Marathe, A.E. Condon, R.M. Corn, On combinatorial DNA word design. J. Comput. Biol. **8**(3), 201–220 (2001)
31. D. Boneh, R.J. Lipton, Making DNA computers error resistant. in *Proceedings of Second Annual DIMACS Conference on DNA computing* 102–110 (1996)
32. R.M. Karp, C. Kenyon, O. Warts, Error-resilient DNA computation. in *Proceedings of Seventh Annual ACM-SIAM Symposium on Discrete Algorithm* 458–467 (1997)