Available online at www.sciencedirect.com
SCIENCE DIRECT®
BioSystems 82 (2005) 203–207
BioSystems
www.elsevier.com/locate/biosystems

Solving maximum cut problems in the Adleman–Lipton model

Dongmei Xiao[a,b], Wenxia Li[a,c,*], Zhizhou Zhang[a], Lin He[a]

[a] *Bio-X DNA Computer Consortium, Shanghai Jiao Tong University, Shanghai 200030, PR China*
[b] *Department of Mathematics, Shanghai Jiao Tong University, Shanghai 200030, PR China*
[c] *Department of Mathematics, East China Normal University, Shanghai 200062, PR China*

Received 18 April 2005; received in revised form 28 June 2005; accepted 28 June 2005

Abstract

In this paper, we consider a procedure for solving maximum cut problems in the Adleman–Lipton model. The procedure works in $O(n^2)$ steps for maximum cut problems of an undirected graph with n vertices.

Keywords: Maximum cut problem; NP-complete problem; Adleman–Lipton model; DNA computing

1. Introduction

In recent works for high-performance computing, computation with DNA molecules, i.e. DNA computing, has considerable attention as one of non-silicon-based computing. Watson–Crick complementarity and massive parallelism are two important features of DNA. Using the features, one can solve an NP-complete problem, which usually needs exponential time on a silicon-based computer, in a polynomial number of steps with DNA molecules. As the first work for DNA computing, Adleman (1994) presented an idea of solving the Hamiltonian path problem of size n in $O(n)$ steps using DNA molecules. Lipton (1995) demonstrated that Adleman's experiment could be used to determine the NP-complete satisfiability (SAT) problem (the first NP-complete problem). Ouyang et al. (1997) presented a molecule biology-based experimental solution to the maximal clique NP-complete problem. In recent years, lots of papers have occurred for designing DNA procedures and algorithms to solve various NP-complete problems. Moreover, procedures for primitive operations, such as logic or arithmetic operations, have been also proposed so as to apply DNA computing on a wide range of problems (Frisco, 2002; Fujiwara et al., 2004; Guarnieri et al., 1996; Gupta et al., 1997; Hug and Schuler, 2001; Kamio et al., 2003).

In this paper, the DNA operations proposed by Adleman (1994) and Lipton (1995) are used for figuring out solutions of *maximum cut NP-complete problems*: for a graph $G = (V, E)$ find a subset C of the vertices that maximizes the number of edges that have one vertex in C and one not in C. It is clear that if C is a solution to the maximum cut problems, then so is

* Corresponding author. Tel.: +86 21 62233060; fax: +86 21 52682621.
E-mail address: wxli@math.ecnu.edu.cn (W. Li).

0303-2647/$ – see front matter © 2005 Elsevier Ireland Ltd. All rights reserved.
doi:10.1016/j.biosystems.2005.06.009

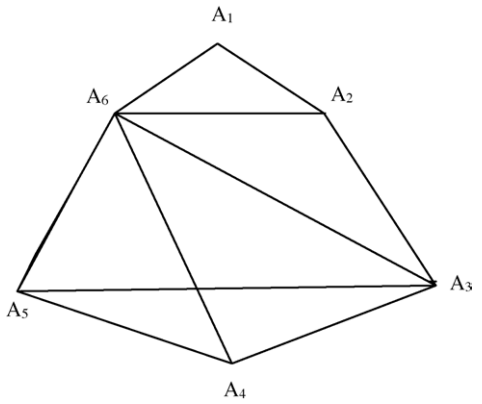

Fig. 1. Graph G.

$V \setminus C$ by the definition. The graph G in Fig. 1 defines such a problem. It is easy to see that $C = \{A_6, A_3\}$ (equivalently $C = \{A_5, A_4, A_2, A_1\}$) is a solution to the maximum cut problem for graph G in Fig. 1.

The rest of this paper is organized as follows. In Section 2, the Adleman–Lipton model is introduced in detail. Section 3 introduces a DNA algorithm for solving the maximum cut problem and the complexity of the proposed algorithm is described. We give conclusions in Section 4.

2. The Adleman-Lipton model

Bio-molecular computers work at the molecular level. Because biological and mathematical operations have some similarities, DNA, the genetic material that encodes for living organisms, is stable and predictable in its reactions and can be used to encode information for mathematical systems.

DNA is the major information storage molecule in living cells, and billions of years of evolution have tested and refined both this wonderful informational molecule and highly special enzymes that can either duplicate the information in DNA molecules or transmit this information to other DNA molecules.

A DNA (deoxyribonucleic acid) is a polymer, which is strung together from monomers called deoxyribonucleotides (Păun et al., 1998). Distinct nucleotides are detected only with their bases. Those bases are, respectively, abbreviated as adenine (A) , guanine (G), cytosine (C), and thymine (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, e.g. the singled strands 5′-ACCTGGATGTAA-3′ and 3′-TGGACCTACATT-5′ can form a double strand. We also call the strand 3′-TGGACCTACATT-5′ as the complementary strand of 5′-ACCTGGATGTAA-3′ and simply denote 3′-TGGACCTACATT-5′ by $\overline{\text{ACCTGGATGTAA}}$. The length of a single stranded DNA is the number of nucleotides comprising the single strand. Thus, if a single stranded DNA includes 20 nucleotides, it is called a 20 mer. The length of a double stranded DNA (where each nucleotide is base paired) is counted in the number of base pairs. Thus, if we make a double stranded DNA from a single stranded 20 mer, then the length of the double stranded DNA is 20 base pairs, also written as 20 bp.

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

(1) *Merge* (T_1, T_2): for two given test tubes T_1, T_2 it stores the union $T_1 \cup T_2$ in T_1 and leaves T_2 empty;
(2) *Copy* (T_1, T_2): for a given test tube T_1 it produces a test tube T_2 with the same contents as T_1;
(3) *Detect* (T): given a test tube T it outputs "yes" if T contains at least one strand, otherwise, outputs "no";
(4) *Separation* (T_1, X, T_2): for a given test tube T_1 and a given set of strings X it removes all single strands containing a string in X from T_1, and produces a test tube T_2 with the removed strands;
(5) *Selection* (T_1, L, T_2): for a given test tube T_1 and a given integer L it removes all strands with length L from T_1, and produces a test tube T_2 with the removed strands;
(6) *Cleavage* $(T, \sigma_0\sigma_1)$: for a given test tube T and a string of two (specified) symbols $\sigma_0\sigma_1$ it cuts each double strand containing $\begin{bmatrix} \sigma_0\sigma_1 \\ \overline{\sigma_0\sigma_1} \end{bmatrix}$ in T into two double strands as follows:

$$\begin{bmatrix} \alpha_0\sigma_0\sigma_1\beta_0 \\ \alpha_1\overline{\sigma_0\sigma_1}\beta_1 \end{bmatrix} \Longrightarrow \begin{bmatrix} \alpha_0\sigma_0 \\ \alpha_1\overline{\sigma_0} \end{bmatrix}, \begin{bmatrix} \sigma_1\beta_0 \\ \overline{\sigma_1}\beta_1 \end{bmatrix}$$

(7) *Annealing* (T): for a given test tube T it produces all feasible double strands in T. The produced double strands are still stored in T after *annealing*;
(8) *Denaturation* (T): for a given test tube T it dissociates each double strand in T into two single strands;
(9) *Discard* (T): for a given test tube T it discards the tube T;
(10) *Append* (T, Z): for a given test tube T and a given short DNA singled strand Z it appends Z onto the end of every strand in the tube T.

Since these ten manipulations are implemented with a constant number of biological steps for DNA strands (Păun et al., 1998), we assume that the complexity of each manipulation is O(1) steps.

3. DNA algorithm for the maximum cut problem

Let $G = (V, E)$ be a graph with the set of vertices being $V = \{A_k | k = 1, 2, \ldots, n\}$ and the set of edges being $E = \{e_{i,j} |$ for some $n \geq i > j \geq 1\}$. Let $|E| = s$. Then $s \leq \frac{1}{2}n(n-1)$.

In the following, the symbols 0, 1, #, A_k, and B_k ($k = 1, 2, \cdots, n$) denote distinct DNA singled strands with same length, say 10-mer. Obviously the length of the DNA singled strands greatly depends on the size of the problem involved in order to distinguish all above symbols and to avoid hairpin formation (Li et al., 2003). We choose DNA singled strands $y_{i,j}$ to encode the edges connecting the vertices A_i and A_j with length of 10-mer. All these $y_{i,j}$ can be taken the same, say y_1, for our problem. For convenience of argument we still use a dummy symbol $y_{i,j}$ of length 0-mer if the vertices A_i and A_j is not connected by an edge or $i = j$. Let

$$P = \{0, 1, A_1\#, \#B_n, A_kB_{k-1} | k = 2, 3, \ldots, n\},$$

$$Q = \{\overline{\#}, \overline{B_k 1 A_k}, \overline{B_k 0 A_k} | k = 1, 2, \ldots, n\}, \quad \text{and}$$

$$H = \{y_{i,j} \mid i, j = 1, 2, \ldots, n\}.$$

We design the following algorithm to solve the maximum cut problems and give the corresponding DNA operations as follows:

(1) For a graph with n vertices, each possible subset of the set V of vertices is represented by an n-digit binary number. A bit set to 1 represents a vertex in the subset, and a bit set to 0 represents a vertex out of the subset. For example, the subset (A_6, A_5) in Fig. 1 is represented by the binary number 110000. In this way, we transform all possible subsets of V in an n-vertex graph into an ensemble of all n-digit binary numbers. We call this the data pool.
(1-1) *Merge* (P, Q);
(1-2) *Annealing* (P);
(1-3) *Denaturation* (P);
(1-4) *Separation* (P, $\{A_1\#\}$, T_{tmp});
(1-5) *Discard* (P);
(1-6) *Separation* (T_{tmp}, $\{\#B_n\}$, P).
After above six steps of manipulation, singled strands in tube P will encode all 2^n subsets of V in the form of n-digit binary numbers. For example, for the graph in Fig. 1 with $n = 6$ we have, e.g. the singled strand $\#B_6 1 A_6 B_5 1 A_5 B_4 1 A_4 B_3 1 A_3 B_2 0 A_2 B_1 0 A_1\# \in P$, which denotes the subset $\{A_6, A_5, A_4, A_3\}$ corresponding to the binary number 111100. This operation can be finished in O(1) steps since each manipulation above works in O(1) steps.
(2) For each element in the data pool that represents some subset C of V, we count the number of edges that have one vertex in C and one not in C. Let the subset C correspond the n-digit binary number $a_n \cdots a_i \cdots a_j \cdots a_1$. For each pair (a_i, a_j) with $a_i = 1, a_j = 0$ or $a_i = 0, a_j = 1$ we append the singled strand $y_{i,j}$ or $y_{j,i}$ to the end of the singled strand which encode the n-digit binary number $a_n \cdots a_i \cdots a_j \cdots a_1$. For example, the singled strands $\#B_6 1 A_6 B_5 1 A_5 B_4 1 A_4 B_3 1 A_3 B_2 0 A_2 B_1 0 A_1\#$ (representing the binary number 111100 for the graph in Fig. 1) is transformed into $\#B_6 1 A_6 B_5 1 A_5 B_4 1 A_4 B_3 1 A_3 B_2 0 A_2 B_1 0 A_1 \# y_{6,2}$ $y_{6,1} y_{3,2}$ where the singled strands $y_{5,2}$, $y_{5,1}$, $y_{4,2}$, $y_{4,1}$, $y_{3,1}$ do not appear since there are not corresponding edges in the graph shown in Fig. 1 and so they all have length 0-mer by the definition of $y_{i,j}$. It means that if we take the subset C of vertices of the graph in Fig. 1 to be $\{A_6, A_5, A_4, A_3\}$, then there are totally three edges A_6A_2, A_6A_1, A_3A_2 which have one vertex in C and one not in C.

For $k = n$ to $k = 1$

(2-1) *Separation* $(P, \{B_k 1 A_k\}, T_1)$

For $i = n$ to $i = 1$

(2-2) *Separation* $(T_1, \{B_i 0 A_i\}, T_2)$

(2-3) *Append* $(T_2, y_{k,i})$

(2-4) *Merge* (T_1, T_2)

End For

(2-5) *Merge*(P, T_1)

End For

In the above operation we use two "For" clauses. Thus this operation can be finished in $O(n^2)$ steps since each single manipulation above works in O(1) steps.

(3) We take out those singled strands in P with largest length, which give the solutions to maximum cut problems. For example, for the graph in Fig. 1, those singled strands in P with largest length are $\#B_6 1 A_6 B_5 0 A_5 B_4 0 A_4 B_3 1 A_3 B_2 0 A_2 B_1 0 A_1 \# y_{6,5} y_{6,4} y_{6,2} y_{6,1} y_{3,5} y_{3,4} y_{3,2}$ and $\#B_6 0 A_6 B_5 1 A_5 B_4 1 A_4 B_3 0 A_3 B_2 1 A_2 B_1 1 A_1 \# y_{5,6} y_{5,3} y_{4,6} y_{4,3} y_{2,6} y_{2,3} y_{1,6}$. Therefore, solutions to maximum cut problems for the graph in Fig. 1 are $\{A_6, A_3\}$ and $\{A_5, A_4, A_2, A_1\}$ in each of which there are seven edges to have only one vertex in the resulting subsets.

For $k = s$ to $k = 1$

(3-1) *Selection* $(P, 30n + 20 + 10k, T)$

(3-2) If *Detect* (T) is "yes", then End For and k is the number of edges corresponding to maximum cuts C, else continue the circulation.

Since s is the size of the set E of edges and $s \leq \frac{1}{2}n(n-1)$, the above "For" clause can be finished in $O(n^2)$ steps.

(4) Finally the *Read* operation is applied to giving the exact solutions to the maximum cut problems. For example, for the graph in Fig. 1, the maximum cuts are $\{A_6, A_3\}$ and $\{A_5, A_4, A_2, A_1\}$. This operation works in O(1) steps.

(4-1) *Read* (T).

Finally, from above four steps of operations we obtain:

Theorem 3.1. *The solutions of maximum cut problems for a graph with n vertices can be figured out in $O(n^2)$ steps using DNA molecules.*

Proof. As the algorithm described above, the solutions of a maximum cut problem for a graph with n vertices can be implemented in four operations using DNA molecules. The first and fourth operations costs O(1) steps, while the second and third operations costs $O(n^2)$ steps. Thus, the solutions of maximum cut problems for a graph with n vertices can be figured out in $O(n^2)$ steps using DNA molecules. □

4. Conclusions

As the first work for DNA computing, (Adleman, 1994) presented an idea to demonstrate that deoxyribonucleic acid (DNA) strands can be applied to solving the Hamiltonian path NP-complete problem of size n in $O(n)$ steps using DNA molecules. Adleman's work shows that one can solve an NP-complete problem, which usually needs exponential time on a silicon-based computer, in a polynomial number of steps with DNA molecules. From then on, Lipton (1995) demonstrated that Adleman's experiment could be used to determine the NP-complete satisfiability (SAT) problem (the first NP-complete problem). Ouyang et al. (1997) showed that restriction enzymes could be used to solve the NP-complete clique problem. In recent years, lots of papers have occurred for designing DNA procedures and algorithms to solve various NP-complete problems. As Guo et al. (2005) pointed out, it is still important to design DNA procedures and algorithms for solving various NP-complete problems since it is very difficult to use biological operations for replacing mathematical operations.

In this paper, we propose a procedure for solving maximum cut NP-complete problems in the Adleman–Lipton model. The procedure works in $O(n^2)$ steps for maximum cut problems of an undirected graph with n vertices. All our results in this paper are based on a theoretical model. However, the proposed procedures can be implemented practically since every DNA manipulation used in this model has been already realized in lab level.

Acknowledgments

This was supported by Bio-X DNA Computer Consortium No. 03DZ14025. The author Wenxia Li was supported by National Science Foundation of China #10371043.

References

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

Frisco, P., 2002. Parallel arithmetic with splicing. Romanian J. Inf. Sci. Technol. 2, 113–128.

Fujiwara, A., Matsumoto, K., Chen, Wei, 2004. Procedures for logic and arithmetic operations with DNA molecules. Int. J. Found. Comput. Sci. 15, 461–474.

Guarnieri, F., Fliss, M., Bancroft, C., 1996. Making DNA add. Science 273, 220–223.

Guo, M.Y., Chang, W.L., Ho, M., Lu, J., Cao, J.N., 2005. Is optimal solution of every NP-complete or NP-hard problem determined from its characteristic for DNA-Based computing. BioSystem 80, 71–82.

Gupta, V., Parthasarathy, S., Zaki, M.J., 1997. Arithmetic and logic operations with DNA. In: Proceedings of Third DIMACS Workshop on DNA-Based Computers, 212–220.

Hug, H., Schuler, R., 2001. DNA-based parallel computation of simple arithmetic. In: Proceedings of the Seventh Internatioanl Meeting on DNA-based Computers, 159–166.

Kamio, S., Takehara, A., Fujiwara, A., 2003. Procedures for computing the maximum with DNA strands. in: Arabnia, Humid, R., Mun, Youngsong (Eds.), In: Proceedings of the International Conference on DNA-Based Computers.

Li, D., Huang, H., Li, X., Li, X., 2003. Hairpin formation in DNA computation presents limits for large NP-complete problems. BioSystem 72, 203–207.

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

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

Păun, G., Rozeberg, G., Salomaa, A., 1998. DNA Computing. Springer-Verlag.