

A procedure for MAX-SAT with DNA strands

Yuichiro Tokumaru, Akihiro Fujiwara

Department of Computer Science and Electronics
Kyushu Institute of Technology
680-4 Kawazu, Iizuka, Fukuoka 820-8502, JAPAN
E-mail: fujiwara@cse.kyutech.ac.jp

Abstract: *In recent works for high performance computing, computation with DNA molecules, that is, DNA computing, has had considerable attention as one of non-silicon based computing. Using features of DNA molecules, we can solve some NP optimization problems, which usually need exponential time on a silicon based computer, in a polynomial number of steps with DNA molecules. In this paper, we propose a procedure for MAX-SAT, which is a well-known NP-hard problem. An input of the problem consists of n Boolean variables and a Boolean function which has h clauses, and each coefficient of the Boolean function is denoted by a binary number of m bits. For MAX-SAT, we propose a procedure which runs in $O(1)$ steps using $O(hm2^n)$ DNA strands. Since an input of the procedure is prepared in $O(n + m)$ steps using $O(hm2^n)$ DNA strands, we solve MAX-SAT in $O(n + m)$ steps using $O(hm2^n)$ DNA strands.*

Keywords: DNA computing, MAX-SAT

1 Introduction

In recent works for high performance computing, computation with DNA molecules, that is, DNA computing, has had considerable attention as one of non-silicon based computing. DNA molecules have two important features, which are Watson-Crick complementarity and massive parallelism. Using the features, we can solve an NP optimization problems, which usually need exponential time on a silicon based computer, in a polynomial number of steps with DNA molecules. As the first work for DNA computing, Adleman [1] presented an idea of solving the Hamiltonian path problem of size n in $O(n)$ steps using DNA molecules. The proposed idea was successfully tested in a lab experiment for a small graph. There are a number of other works with DNA molecules for combinatorial NP optimization problems [2, 3, 7, 8, 13].

In this paper, we propose a procedure for MAX-SAT, which is a well-known NP-hard problem. The MAX-SAT is a fundamental problem in operations research, and plays important role in computer science. Inputs of the problem consist of n Boolean variables and a Boolean function which has h clauses. We assume that each coefficient of the Boolean function is denoted by a binary number of m bits. An output of the problem is an assignment to the variables which maximize the number of satisfied clauses. For MAX-SAT, we propose a procedure which performs an exhaustive search for all solutions using the feature of DNA molecules. The procedure runs in $O(1)$ steps using $O(hm2^n)$ DNA strands. Since an input of the procedure is prepared in $O(n + m)$ steps using $O(hm2^n)$ DNA strands, we solve MAX-SAT in $O(n + m)$ steps using $O(hm2^n)$ DNA strands.

2 Preliminaries

2.1 Computational model for DNA computing

A number of theoretical or practical computational models have been proposed for DNA computing [2, 5, 6, 7, 10, 11, 12]. A computational model used in this paper is the same model as [4]. We briefly introduce the model in this subsection.

A *single strand* of DNA is defined as a string of symbols over a finite alphabet Σ . We define the alphabet $\Sigma = \{\sigma_0, \sigma_1, \dots, \sigma_{m-1}, \overline{\sigma_0}, \overline{\sigma_1}, \dots, \overline{\sigma_{m-1}}\}$. In the alphabet, the symbols $\sigma_i, \overline{\sigma_i}$ ($0 \leq i \leq m - 1$) are *complements*.

Two single strands can form a *double strand* if and only if the single strands are complements of each other. A double strand with $\sigma_i, \overline{\sigma_i}$ is denoted by $\begin{bmatrix} \sigma_i \\ \overline{\sigma_i} \end{bmatrix}$.

The single and double strands are stored in *test tubes*. For example, $T_1 = \{\sigma_0\sigma_1, \overline{\sigma_0\sigma_1}\}$ denotes a test tube which contains two kinds of single strands $\sigma_0\sigma_1$ and $\overline{\sigma_0\sigma_1}$.

Using the DNA strands, the following nine manipulations are allowed on the computational model. Since these nine manipulations are implemented with a constant number of biological steps for DNA strands [9], we assume that the complexity of each manipulation is $O(1)$ steps. (See [4] for more details.)

1. *Merge* : Given two test tubes T_1, T_2 , $Merge(T_1, T_2)$ stores the union $T_1 \cup T_2$ in T_1 .
2. *Copy* : Given a test tube T_1 , $Copy(T_1, T_2)$ produces a test tube T_2 with the same contents as T_1 .
3. *Detect* : Given a test tube T , $Detect(T)$ outputs “yes” if T contains at least one strand, otherwise, $Detect(T)$ outputs “no”.
4. *Separation* : Given a test tube T_1 and a set of strings X , $Separation(T_1, X, T_2)$ 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* : Given a test tube T_1 and an integer L , $Selection(T_1, L, T_2)$ removes all strands, whose length is L , from T_1 and produces a test tube T_2 with the removed strand. (The length of a strand is the number of symbols in the strand.)
6. *Cleavage* : Given a test tube T and a string of two symbols $\sigma_0\sigma_1$, $Cleavage(T, \sigma_0\sigma_1)$ cuts each double strand containing $\begin{bmatrix} \sigma_0\sigma_1 \\ \sigma_0\sigma_1 \end{bmatrix}$ in T into two double strands as follows.
$$\begin{bmatrix} \alpha_0\sigma_0\sigma_1\beta_0 \\ \alpha_0\overline{\sigma_0\sigma_1}\beta_0 \end{bmatrix} \Rightarrow \begin{bmatrix} \alpha_0\sigma_0 \\ \alpha_0\overline{\sigma_0} \end{bmatrix}, \begin{bmatrix} \sigma_1\beta_0 \\ \overline{\sigma_1}\beta_0 \end{bmatrix}$$
7. *Annealing* : Given a test tube T , $Annealing(T)$ produces all feasible double strands from single strands in T . (The produced double strands are still stored in T after *Annealing*.)
8. *Denaturation* : Given a test tube T , $Denaturation(T)$ dissociates each double strand in T into two single strands.
9. *Empty* * : Given a test tube T , $Empty(T)$ sets $T = \phi$

2.2 Representation of binary numbers with DNA strands

In this subsection, we define the representation of binary numbers with DNA strands. We first define the alphabet which is a set of component of each DNA strand.

$$\Sigma = \{A_0, A_1, \dots, A_{n-1}, B_0, B_1, \dots, B_{m-1}, C_0, C_1, D_0, D_1, 0, 1, \#, \overline{A_0}, \overline{A_1}, \dots, \overline{A_{n-1}}, \overline{B_0}, \overline{B_1}, \dots, \overline{B_{m-1}}, \overline{C_0}, \overline{C_1}, \overline{D_0}, \overline{D_1}, \overline{0}, \overline{1}, \overline{\#}\}$$

In the above alphabet, A_0, A_1, \dots, A_{n-1} denote addresses of numbers, and B_0, B_1, \dots, B_{m-1} denote bit positions. C_0, C_1, D_0, D_1 are specified symbols cut by *Cleavage*. A symbol “#” is a special symbol for *Separation*.

Using the above alphabet, a value of a bit is represented by a single strand defined as follows.

$$S_{i,j} = D_1 A_i B_j C_0 C_1 V_{i,j} D_0$$

In the above single strand, $V_{i,j} = 0$ if a value of the bit is 0, otherwise $V_{i,j} = 1$. We call the single strand $S_{i,j}$ a *memory strand* [4], and a binary number stored in an address i is represented by a set of memory strands

* $Empty(T)$: is equivalent to $Copy(\phi, T)$.

$\{S_{i,m-1}, S_{i,m-2}, \dots, S_{i,0}\}$, which denote binary bits $x_{m-1}, x_{m-2}, \dots, x_0$ respectively. We assume that V_i denotes the number stored in an address i as follows.

$$V_i = \sum_{j=0}^{m-1} V_{i,j} \times 2^j$$

In the following, $A_{x,y}$ denotes address part A_i in the memory strands. In the description, $A_{x,y} = A_i$ means higher and lower parts of i is x and y , respectively.

3 Definition of MAX-SAT using DNA strands

3.1 Definition of MAX-SAT

MAX-SAT is a problem which requests an assignment to Boolean variables which maximizes the number of satisfied clauses of a given Boolean function. The MAX-SAT is defined as follows.

Input A Boolean function F which has n Boolean variables and h clauses.

Output An assignment to Boolean variables which maximizes a number of satisfied clauses, and the number of satisfied clauses.

We show an example of an input Boolean function, which has 4 clauses ($h = 4$) and 3 variables ($n = 3$) in the following.

$$F(x_0, x_1, x_2) = (x_0 + x_1)(x_0 + \overline{x_1})(x_1 + x_2)(x_2)$$

In this case, assignments $(x_2, x_1, x_0) = (1, 0, 1)$ and $(x_2, x_1, x_0) = (1, 1, 1)$ are outputs of the MAX-SAT which maximize a number of satisfied clauses. The number of satisfied clauses is 4.

3.2 Data representation of MAX-SAT

Using the memory strands defined in Section 2, an input of the MAX-SAT is denoted as follows.

$$Q_{i,1,k} = D_1 A_{i,1} B_k C_0 C_1 V_{i,1,k} D_0 \quad (0 \leq i \leq h-1, 0 \leq k \leq m-1)$$

The memory strands $Q_{i,1,k}$ denotes coefficients of the i -th clause. In case of $0 \leq k \leq n-1$, $V_{i,1,k} = 1$ if and only if x_k is in the i -th clause. In case of $n \leq k \leq 2n-1$, $V_{i,1,k} = 1$ if and only if $\overline{x_{k-n}}$ is in the i -th clause. For example, let $(x_0 + \overline{x_1})$ be i -th clause. Then, the following memory strands denote the clause.

$$\begin{aligned} Q_{i,1,5} &= D_1 A_{i,0} B_2 C_0 C_1 0 D_0, & Q_{i,1,4} &= D_1 A_{i,0} B_1 C_0 C_1 1 D_0, & Q_{i,1,3} &= D_1 A_{i,0} B_0 C_0 C_1 0 D_0, \\ Q_{i,1,2} &= D_1 A_{i,0} B_2 C_0 C_1 0 D_0, & Q_{i,1,1} &= D_1 A_{i,0} B_1 C_0 C_1 0 D_0, & Q_{i,1,0} &= D_1 A_{i,0} B_0 C_0 C_1 1 D_0 \end{aligned}$$

In addition, the following four kinds of memory strands are prepared before the procedure. (The memory strands are used in the procedure.)

(1) Memory strands which denote k -th variable x_k of i -th assignment:

$$P_{i,0,k} = D_1 A_{i,0} B_k C_0 C_1 V_{i,0,k} D_0 \quad (0 \leq i \leq 2^n - 1, 0 \leq k \leq m-1)$$

For example, let $(x_2, x_1, x_0) = (1, 0, 1)$ be i -th assignment. Then, the following memory strands denote the assignment.

$$P_{i,0,2} = D_1 A_{i,0} B_2 C_0 C_1 1 D_0, P_{i,0,1} = D_1 A_{i,0} B_1 C_0 C_1 0 D_0, P_{i,0,0} = D_1 A_{i,0} B_0 C_0 C_1 1 D_0$$

We also assume that the following memory strand denotes negation of k -th variable of i -th assignment.

$$P_{i,0,n+k} = D_1 A_{i,0} B_k C_0 C_1 \overline{V_{i,0,k}} D_0 \quad (0 \leq i \leq 2^n - 1, 0 \leq k \leq m-1)$$

(2) Memory strands which denote a Boolean value of k -th clause by i -th assignment:

$$Y_{i,0,k} = D_1 A_{i,0} B_k C_0 C_1 V_{i,0,k} D_0 \quad (0 \leq i \leq 2^n - 1, 0 \leq k \leq h - 1)$$

In the above $Y_{i,0,k}, V_{i,0,k} = 1$ if and only if a Boolean value of k -th clause is 1 by i -th assignment.

(3) Memory strands which denote the number of satisfied clauses by i -th assignment:

$$Z_{i,j,k} = D_1 A_{i,j} B_k C_0 C_1 V_{i,j,k} D_0 \quad (0 \leq i \leq 2^n - 1, 0 \leq j \leq h, 0 \leq k \leq m - 1)$$

The memory strands $Z_{i,j,m-1}, Z_{i,j,m-2}, \dots, Z_{i,j,0}$ denote a binary number. (Details of the memory strands are described later.)

(4) Memory strands which denote the maximum of the numbers of satisfied clauses:

$$M_{2^n,0,k} = D_1 A_{2^n,0} B_k C_0 C_1 V_{2^n,0,k} D_0 \quad (0 \leq k \leq m - 1)$$

The memory strands $M_{2^n,0,m-1}, M_{2^n,0,m-2}, \dots, M_{2^n,0,0}$ denote an output value of MAX-SAT.

For the above memory strands, we prove the following lemma. (Proof of the lemma is omitted due to space limitation.)

Lemma 1 *The input and output memory strands for MAX-SAT can be prepared in $O(m+n)$ steps using $O(hm2^n)$ DNA strands. \square*

Using the above memory strands, input and output test tubes of the procedure for MAX-SAT are defined as follows.

Input: $T_{input,p} = \{Q_{p,1,k} \mid 0 \leq k \leq m - 1\}$ ($0 \leq p \leq h - 1$)

Output: $T_{output} = \{M_{2^n,0,k}, P_{i,0,k} \mid i \in \{0, 1, \dots, 2^n - 1\}, 0 \leq k \leq m - 1\}$

4 A procedure for MAX-SAT using DNA strands

The procedure which solves MAX-SAT mainly consists of the following three steps.

Step 1 Compute a Boolean value of each clause for each assignment.

Step 2 Sum results in Step 1 for each assignment, and compute the number of satisfied clauses for each assignment.

Step 3 Compute the maximum number of satisfied clauses from result of Step 2, and find assignment which maximizes the number of satisfied clauses.

We next describe the procedure using DNA strands. First of all, initial states and the roles of the test tubes used in the procedure are shown below. In the following, every p is assumed to be $0 \leq p \leq h - 1$.

$T_{input,p}$: An input test tube given above.

$T_{ans,p}$: A test tube with DNA strands which denote all assignments to Boolean variables $P_{i,0,p}$ ($0 \leq i \leq 2^n - 1$) for p -th clauses.

$T_{tmp,p}, T'_{tmp,p}$: Test tubes which are temporarily used in the procedure.

$T_{value,p}$: A test tube with memory strands which denote Boolean values $Y_{i,0,p}$ ($0 \leq i \leq 2^n - 1$) of p -th clause.

$T_{valueall}$: A test tube with memory strands which denote Boolean values of $Y_{i,0,k}$ ($0 \leq i \leq 2^n - 1, 0 \leq k \leq h - 1$) of all clauses.

T_1 : A test tube with memory strands whose values are 1.

T_{con1}, T_{con2} : Test tubes with DNA strands which connect memory strands.

T_{num}, T'_{num} : Test tubes with DNA strands which store candidates of the numbers of satisfied clauses $Z_{i,j,k}$ ($0 \leq i \leq 2^n - 1, 0 \leq j \leq h, 0 \leq k \leq m - 1$).

T_{output} : An output test tube given above.

We now show an overview of the procedure. (We omit details of the procedure due to space limitation.)

Procedure MAX-SAT

Step 1 Compute each Boolean value $Y_{i,0,k}$ ($0 \leq i \leq 2^n - 1, 0 \leq k \leq h - 1$) of each clause for all assignments $P_{i,0,k}$ ($0 \leq i \leq 2^n - 1, 0 \leq k \leq m - 1$).

To execute Step 1, following substeps (1-1) ~ (1-6) are performed in parallel for all clauses.

(1-1) Extract single strands which denote coefficients of clauses $Q_{i,1,k}$ ($0 \leq i \leq h - 1, 0 \leq k \leq m - 1$) and variable assignments $P_{j,0,k}$ ($0 \leq j \leq 2^n - 1$) from test tubes $T_{input,p}, T_{ans,p}$ ($0 \leq p \leq h - 1$), and store the extracted single strands into test tube $T_{tmp,p}$.

(1-2) Execute a logic operation $P_{i,0,k} = P_{i,0,k} \cap Q_{p,1,k}$ ($0 \leq i \leq 2^n - 1, 0 \leq k \leq m - 1$) for test tubes $T_{tmp,p}$ ($0 \leq p \leq h - 1$) in parallel.

(1-3) First, extract $P_{i,0,k}$ ($0 \leq i \leq 2^n - 1, 0 \leq k \leq m - 1$) from each test tube $T_{tmp,p}$ ($0 \leq p \leq h - 1$), store the extracted strands into $T'_{tmp,p}$, and then, empty the test tube $T_{tmp,p}$. Next, extract the memory strands, whose value is 1, from test tubes $T'_{tmp,p}$ ($0 \leq p \leq h - 1$), store the extracted strands into $T_{tmp,p}$, and then, empty the test tube $T'_{tmp,p}$.

(1-4) Extract memory strands $Y_{i,0,p}$ ($i \in \{0, 1, \dots, 2^n - 1\}$), which denote Boolean values of the clauses corresponding to the address of the memory strand in $T_{tmp,p}$, from test tubes $T_{value,p}$ ($0 \leq p \leq h - 1$), and store the extracted strands into $T_{tmp,p}$.

(1-5) Set values of memory strands $Y_{i,0,p}$ in test tubes $T_{tmp,p}$ ($0 \leq p \leq h - 1$) to 1, and return the memory strands to each $T_{value,p}$ ($0 \leq p \leq h - 1$).

(1-6) Extract memory strands $Y_{i,0,p}$ ($0 \leq i \leq 2^n - 1$) from test tube $T_{value,p}$ ($0 \leq p \leq h - 1$), and store the extracted memory strands into the test tube $T_{valueall}$.

After Step 1, single strands $Y_{i,0,k}$ ($0 \leq i \leq 2^n - 1, 0 \leq k \leq h - 1$), which denote Boolean values for all assignments, are stored in a test tube $T_{valueall}$.

Step 2 In Step 2, we compute the number of satisfied clauses $Z_{i,j,k}$ ($0 \leq j \leq h$) for each assignment $P_{i,0,k}$ ($0 \leq i \leq 2^n - 1, 0 \leq k \leq m - 1$) in parallel. Since Boolean values $Y_{i,0,k}$ for all assignments are obtained in Step 1, we obtain the sum of the number of satisfied clauses if we sum the number of Boolean values for each assignment. In the procedure, we compute the sum by connecting memory strands, whose value is 1, for each address in ascending order. For example, we assume that the following set is a result of Step 1.

$$\{Y_{i,0,0}(1), Y_{i,0,1}(1), Y_{i,0,2}(0), Y_{i,0,3}(1)\}$$

For the above set of single strands, we connect the memory strands in ascending order. Then, we obtain the following set of single strands.

$$\{Y_{i,0,0}(1)Y_{i,0,3}(1), Y_{i,0,0}(1)Y_{i,0,1}(1)Y_{i,0,3}(1)\}$$

In the set of single strands, a length of the longest single strand denotes the maximum number of “1” for all assignments. However, we cannot compute the number directly using $O(1)$ manipulations.

To obtain the number, we use additional single strands, which denote the number. The following single strands are examples in case of $h = 4$.

$$\left\{ \begin{array}{l} \boxed{\alpha \quad \alpha \quad \alpha \quad \alpha} Z_{i,4,0}(0) Z_{i,4,1}(0) Z_{i,4,2}(0), \\ \quad \boxed{\alpha \quad \alpha \quad \alpha} Z_{i,3,0}(1) Z_{i,3,1}(0) Z_{i,3,2}(0), \\ \quad \quad \boxed{\alpha \quad \alpha} Z_{i,2,0}(0) Z_{i,2,1}(1) Z_{i,2,2}(0), \\ \quad \quad \quad \boxed{\alpha} Z_{i,1,0}(1) Z_{i,1,1}(1) Z_{i,1,2}(0), \\ \quad \quad \quad \quad Z_{i,0,0}(0) Z_{i,0,1}(0) Z_{i,0,2}(1) \end{array} \right\}$$

Each of these single strands consists of two parts. First part consists of dummy strands which make length of the single strand a constant, and second part consists of the memory strands which denote the length of the single strand as a binary number. For example, we obtain the following single strands from the above two kinds of single strands.

$$\left\{ \begin{array}{l} \boxed{\alpha \quad \alpha \quad \alpha \quad \alpha} Z_{i,4,0}(0) Z_{i,4,1}(0) Z_{i,4,2}(0), \\ Y_{i,0,0}(1) Y_{i,0,3}(1) \boxed{\alpha \quad \alpha} Z_{i,2,0}(0) Z_{i,2,1}(1) Z_{i,2,2}(0), \\ Y_{i,0,0}(1) Y_{i,0,1}(1) Y_{i,0,3}(1) \boxed{\alpha} Z_{i,1,0}(1) Z_{i,1,1}(1) Z_{i,1,2}(0) \end{array} \right\}$$

As described above, the number of connected memory strands is denoted by additional single strands as a binary number.

Step 2 is realized by performing the following substeps (2-1) ~ (2-4).

- (2-1) Extract memory strands whose values are 1 from test tube $T_{valueall}$, and store the extracted memory strands into a test tube T_1 . Then move strands in T_1 into another test tube T_{con1} .
- (2-2) Execute *Annealing* for strands in T_{con1} , and move the obtained strands into another test tube T_{con2}
- (2-3) Execute *Annealing* for strands in T_{con2} . Then, extract DNA strand of specified length $(\alpha \times h + \beta)$ from T_{con2} , and store the extracted strands into a test tube $T_{tmp,1}$. (α denotes length of a memory strand, and β denotes length of the second part of the additional single strand.)
- (2-4) Extract memory strands $Z_{i,j,k}$ ($0 \leq i \leq 2^n - 1, 0 \leq j \leq h, 0 \leq k \leq m - 1$) which denote length of single strands from the test tube $T_{tmp,1}$, and store the extracted strands into a test tube T_{num} .

Step 3 Compute the maximum number of satisfied clauses from result of Step 2, and find an assignment which maximizes the number of satisfied clauses. The Step 3 is realized by performing the following substeps (3-1), (3-2).

- (3-1) First, copy a test tube T_{num} to another test tube T'_{num} . Next, merge T'_{num} with $M_{2^n,0,k}$ ($0 \leq k \leq m-1$), and compute the maximum of numbers which are denoted by memory strand $Z_{i,j,k}$ ($0 \leq i \leq 2^n - 1, 0 \leq j \leq h$) in test tube T'_{num} . Finally, store the maximum in $M_{2^n,0,k}$.
- (3-2) Find an assignment which maximizes the number of satisfied clauses from $M_{2^n,0,k}$, and store the assignment into T_{output} .

(End of the procedure)

For the above procedure, we obtain the following theorem. (Details are also omitted.)

Theorem 1 *The procedure MAX-SAT can be executed in $O(1)$ steps using $O(hm2^n)$ kinds of DNA strands. \square*

5 Conclusion

In this paper, we proposed a procedure which solves MAX-SAT with DNA strands. This procedure runs in $O(1)$ steps using $O(hm2^n)$ DNA strands. Since an input of the procedure is prepared in $O(n+m)$ steps using $O(hm2^n)$ DNA strands, we solve MAX-SAT in $O(n+m)$ steps using $O(hm2^n)$ DNA strands. Although our results are based on a theoretical model, the proposed procedures can be implemented practically since each DNA manipulation used in the model has been already realized in laboratory. Therefore, we believe that our results will play an important role in the future DNA computing.

References

- [1] L. M. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266:1021–1024, 1994.
- [2] L. M. Adleman. Computing with DNA. *Scientific American*, 279(2):54–61, 1998.
- [3] A. Baumker and W. Dittrich. Fully dynamic search trees for an extension of the bsp model. In *Proc. 8th Symposium on parallel Algorithms and Architectures*, pages 233–241, 1996.
- [4] A. Fujiwara, K. Matsumoto, and W. Chen. Procedures for logic and arithmetic operations with DNA molecules. *International Journal of Foundations of Computer Science*, 15(3), 2004.
- [5] V. Gupta, S. Parthasarathy, and M. J. Zaki. Arithmetic and logic operations with DNA. In *Proceedings of the 3rd DIMACS Workshop on DNA Based Computers*, pages 212–220, 1997.
- [6] H. Hug and R. Schuler. Dna-based parallel computation of simple arithmetic. In *Proceedings of the 7th International Meeting on DNA Based Computers*, pages 159–166, 2001.
- [7] R. J. Lipton. DNA solution of hard computational problems. *Science*, 268:542–545, 1995.
- [8] Q. ouyang, P.D. Kaplan, S.Liu, and A. Libchaber. Dna solution of the maximal clique problem. *Science*, 278:446–449, 1997.
- [9] G. Păun, G. Rozeberg, and A. Salomaa. *DNA computing*. Springer-Verlag, 1998.
- [10] Z. F. Qiu and M. Lu. Arithmetic and logic operations for DNA computers. In *Proceedings of the Second IASTED International conference on Parallel and Distributed Computing and Networks*, pages 481–486, 1998.
- [11] Z. F. Qiu and M. Lu. Take advantage of the computing power of DNA computers. In *Proceedings of the Third Workshop on Bio-Inspired Solutions to Parallel Processing Problems, IPDPS 2000 Workshops*, pages 570–577, 2000.
- [12] J. H. Reif. Parallel biomolecular computation: Models and simulations. *Algorithmica*, 25(2/3):142–175, 1999.
- [13] H. Yoshida and A. Suyama. Solution to 3-SAT by breadth first search. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 54:pp.9–22, 2000.