Available online at www.sciencedirect.com

BioSystems 81 (2005) 1–9

www.elsevier.com/locate/biosystems

A DNA solution of SAT problem by a modified sticker model

Chia-Ning Yang[a, *], Chang-Biau Yang[b, 1]

[a] *Department of Medical Radiation Technology, I-Shou University, No. 1, Section 1, Hsueh-Cheng Road, Ta-Hsu Hsiang, Kaohsiung 840, Taiwan*

[b] *Department of Computer Science and Engineering, National Sun Yat-sen University, No. 70, Lien Hai Road, Kaohsiung, 804, Taiwan*

Received 15 September 2004; received in revised form 7 January 2005; accepted 12 January 2005

Abstract

Among various DNA computing algorithms, it is very common to create an initial data pool that covers correct and incorrect answers at first place followed by a series of selection process to destroy the incorrect ones. The surviving DNA sequences are read as the solutions to the problem. However, algorithms based on such a brute force search will be limited to the problem size. That is, as the number of parameters in the studied problem grows, eventually the algorithm becomes impossible owing to the tremendous initial data pool size. In this theoretical work, we modify a well-known sticker model to design an algorithm that does not require an initial data pool for SAT problem. We propose to build solution sequences in parts to satisfy one clause in a step, and eventually solve the whole Boolean formula after a number of steps. Accordingly, the size of data pool grows from one sort of molecule to the number of solution assignments. The proposed algorithm is expected to provide a solution to SAT problem and become practical as the problem size scales up.

Keywords: SAT problem; DNA computing; Sticker model

1. Introduction

DNA computing is to solve computational problems by employing molecular biology laboratory techniques to manipulate DNA strings so that the strings encode information of the related algorithm and furthermore the strings carrying answer statements are to be recognized. Adleman (1994) first introduced this idea by solving a computationally hard problem, the directed Hamiltonian path problem. Since then, many groups have worked on different mathematic hard problems with the Adleman-style computing (Faullhammer et al., 2000; Gillmor et al., 2002; Guarnieri et al., 1996; Lipton, 1995; Liu et al., 2000; Ouyang et al., 1997; Roy et al., 2002; Sakamoto et al., 2000). For example, Ouyang's (Ouyang et al., 1997) work on solving the maximal clique problem and Faullhammer's (Faullhammer et al., 2000) RNA

* Corresponding author. Tel.: +886 7 6577711x6814; fax: +886 7 6577056.

E-mail addresses: cnyang@isu.edu.tw (C.-N. Yang), cbyang@cse.nsysu.edu.tw (C.-B. Yang).

[1] Tel.: 886 7 5252000x4300/4333, fax: 886 7 5254301.

solution to chess problems are further demonstrations of molecular-based computations.

Conventionally, DNA computing consists of three major steps: (i) generate a data pool of DNA molecules that represent all possible solutions to the studied problem, (ii) systematically operate a series of biology laboratory techniques to in large exclude the DNA strands that do not satisfy logic constraints of the problem, and (iii) collect the surviving DNA molecules for answer readout process. The second step is a typical data-parallel computation that greatly accelerates a tedious computing process for a hard problem. A highly parallel algorithm lies in the delicate data structure so that the encoded information can be distinguished efficiently without leaving errors to the answer readout.

In this paper, we describe an algorithm to solve the satisfiability (SAT) problem (Cormen et al., 2001). Such a problem has been keenly studied both theoretically and experimentally on the basis of DNA computing (Chen and Ramachandran, 2001; Díaz et al., 2001; Faullhammer et al., 2000; Lipton, 1995; Yoshida and Suyama, 1999), since Lipton's paradigm proposed in 1995 demonstrated the feasibility of applying DNA computer to tackle such an NP-complete problem.

2. Backgrounds

2.1. SAT Problem

The SAT problem is to find assignments of a set of Boolean variables that lead the output of a Boolean formula to be true. Consider a Boolean formula $F = (x^{\mathrm{T}} \vee y^{\mathrm{F}}) \wedge (x^{\mathrm{F}} \vee z^{\mathrm{T}}) \wedge (y^{\mathrm{F}} \vee z^{\mathrm{F}})$, where $\wedge$ and $\vee$ are logic AND and OR operations, respectively. Boolean variables x, y, and z are allowed to range over "true" and "false" values. Notations x^{T}, x^{F}, y^{T}, y^{F}, z^{T}, and z^{F} represent the true and false values of the variables x, y, and z. The truth assignments of (x, y, z) for the above formula are $(x^{\mathrm{T}}, y^{\mathrm{F}}, z^{\mathrm{T}})$, $(x^{\mathrm{F}}, y^{\mathrm{F}}, z^{\mathrm{T}})$, and $(x^{\mathrm{F}}, y^{\mathrm{F}}, z^{\mathrm{F}})$. One way to find these assignments is merely by plugging all the 2^3 possible choices, i.e., $(x^{\mathrm{T}}, y^{\mathrm{T}}, z^{\mathrm{T}})$, $(x^{\mathrm{T}}, y^{\mathrm{T}}, z^{\mathrm{F}})$, $(x^{\mathrm{T}}, y^{\mathrm{F}}, z^{\mathrm{T}})$ $(x^{\mathrm{F}}, y^{\mathrm{T}}, z^{\mathrm{T}})$, $(x^{\mathrm{F}}, y^{\mathrm{F}}, z^{\mathrm{T}})$, and $(x^{\mathrm{F}}, y^{\mathrm{T}}, z^{\mathrm{F}})$ $(x^{\mathrm{T}}, y^{\mathrm{F}}, z^{\mathrm{F}})$, and $(x^{\mathrm{F}}, y^{\mathrm{F}}, z^{\mathrm{F}})$, to the formula and checking the satisfiability. This strategy, which is based on brute force search and surveys all 2^n choices for an SAT problem with n variables, is widely adopted in works by Lipton (1995), Faullhammer et al. (2000), Liu et al. (2000), Braich et al. (2002). For a problem involving many variables, such a strategy certainly requires an initial data pool in an enormous size and brings inefficiency in synthesis and selection processes. In our work, instead of surveying all possible assignment sequences generated in the very beginning, we build the solution sequences in parts to satisfy one clause in one step and gradually satisfy the whole formula after several steps. This way, the size of our data pool grows from one sort of molecule to the number of answer sequences.

2.2. Sticker model

The logic of our algorithm is rooted in the sticker model (Păun et al., 1998), which is based on the paradigm of Watson–Crick complementarity. In short, the model involves a long single memory strand and a number of sticker strands or stickers as indicated in Fig. 1. A memory strand is a single-stranded DNA with n bases. It is divided into k non-overlapping substrands, each of which has m bases, and therefore, $n = km$. Each sticker has m bases and complementary to exactly one of the k substrands in the memory strand. During a course of computation, each substrand is identified as a Boolean variable and is considered "true" or "false" as its corresponding sticker is annealed or not (or the other way around). Usually, a memory strand with k subsections can be applied to solve a problem with k or fewer Boolean variables. A memory complex is the term defined as a memory strand where part of the substrands are annealed by the matching stickers, such that the computational information can be carried in a binary format along the memory complex. Our algorithm adopts the essentiality of the sticker model with modification pointed out later.

3. Methods

We use a simple Boolean formula $F = (x^{\mathrm{T}} \vee y^{\mathrm{F}}) \wedge (x^{\mathrm{F}} \vee z^{\mathrm{T}}) \wedge (y^{\mathrm{F}} \vee z^{\mathrm{F}})$ to bring full details about the proposed method. This is an SAT problem with three literals, three clauses, and two variables in each clause. Complicate SAT problems with more literals and more clauses have been applied to exam our method. Though the obtained results

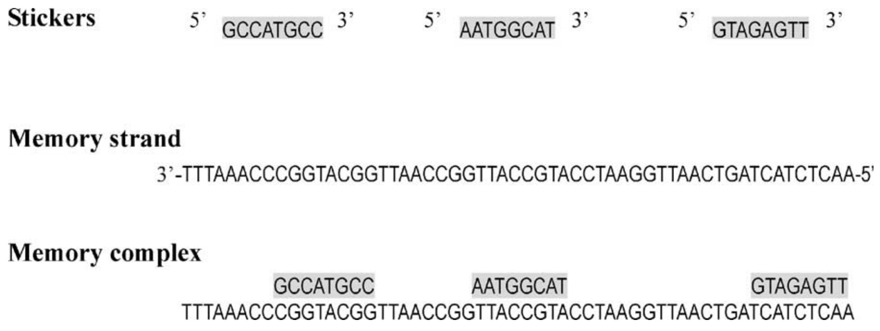

Stickers
5' GCCATGCC 3'
5' AATGGCAT 3'
5' GTAGAGTT 3'
Memory strand
3'-TTTAAACCCGGTACGGTTAACCGGTTACCGTACCTAAGGTTAACTGATCATCTCAA-5'
Memory complex
GCCATGCC
AATGGCAT
GTAGAGTT
TTTAAACCCGGTACGGTTAACCGGTTACCGTACCTAAGGTTAACTGATCATCTCAA

Fig. 1. An example of sticker system, which is encoded information 0101001. Note that the memory strand is composed of a number of eight-basis-long subsections. Each subsection is defined as 0 by annealing no sticker, while the sticker-annealed subsection is defined as 1.

are not provided, they are supportive to the proposed algorithm.

3.1. *Modified sticker model*

For three Boolean variables are involved in the studied formula, we set the memory strand to a 90-bases single-stranded DNA molecule. This memory strand is divided into three 30-base subsections corresponding to the variables *x*, *y* and *z*, as shown in Fig. 2(a). Unlike the conventional sticker model, where each subsection recognizes one sort of sticker, in our model each 30-base subsection recognizes two sorts of stickers, one for true value and one for false value, as indicated in Fig. 2(b). Fig. 2(c) gives examples of how the variable *x* is determined in our modified sticker system. The first 20 bases in the *x*-variable subsection recognize a 20-base sticker x^T (the true value of variable *x*), while the last 20 bases in the same subsection recognize a 20-base sticker x^F

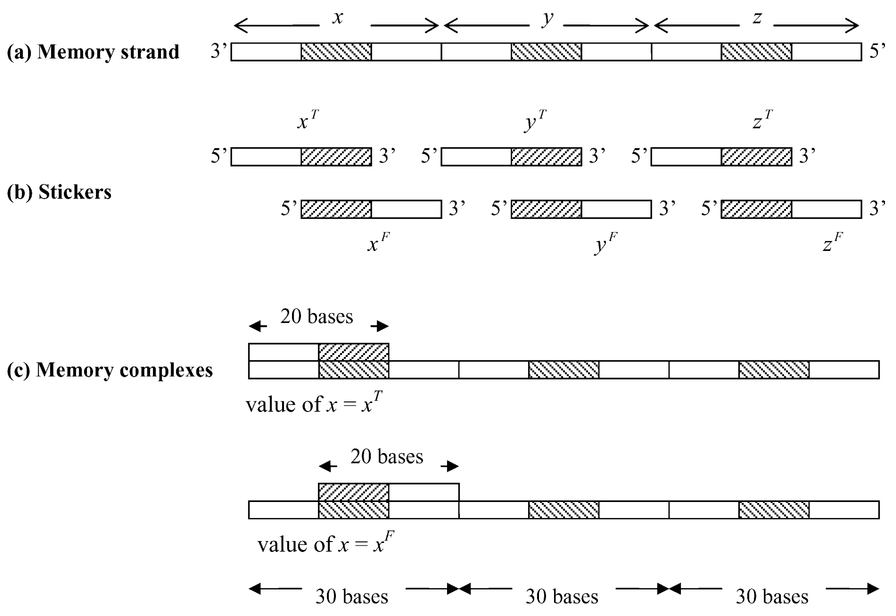


Fig. 2. Illustration of DNA strands involved in the algorithm and examples of value definition in the memory complexes. (a) A 90-base memory strand is divided into three 30-base subsections, each representing a considered variable, *x*, *y*, or *z*. (b) Stickers with true or false value are complementary to subsections of the memory strand. (c) Sickers x^T and x^F are annealed to form memory complexes with true and false values of *x*.

(the false value of variable x). Accordingly, x can be defined as true or false by annealing either sticker x^{T} or x^{F}. Moreover, because each true-false sticker pair share the same sequence in the last 10 bases of a true sticker and the first 10 bases in its corresponding false sticker, the sharing fragment prevents a variable from taking both true and false assignments on one memory complex molecule. Such a design serves for two purposes. First, taking the variable x as an example again, when equal amounts of sticker strands x^{T} and x^{F} are simultaneously in the presence of memory strands or memory complexes with undefined value of x, both stickers x^{T} and x^{F} will randomly anneal to the memory strands. Consequently, roughly one half of the resulted memory complexes will carry the true value and the other half, false value. This function shall be very useful to assign value to x, especially when a logic constraint does not care what the value of x is. Second, for a tube of memory complexes in which the variable x is already defined as true, any additional sticker x^{F} will not redefine the variable x. In the conventional sticker model, a false value, which is usually defined by no sticker annealed to a corresponding site on the memory complex, may be subject to the conversion into a true value by accidentally annealing a matching sticker, if contamination is not preventable. Such an incident could be a source of error in computing. However, our modified model allows the memory complexes to stay unaffected, while some free sticker molecules, with values opposite to the previously defined values on the memory complexes, are in the same tube.

3.2. Algorithm

The fundamental idea of the presented algorithm starts with memory strands where the variables are to be defined by annealing stickers under the circumstance of satisfying the studied SAT formula. Three annealing steps are involved. Each step considers simply one clause and forms memory complexes that satisfy this clause. Gradually, as we move from the first clause to the last clause in the formula, the memory complexes cumulate the meaningful variable definitions and are ready for readout process. The following are the procedures.

Step 1: Assign variables x and y onto memory strands to satisfy the first clause, $(x^{T} \vee y^{F})$; selectively collect the resulted memory complexes, containing stickers x^{T} or y^{F}. At this stage, the collected memory complexes satisfy the first clause.

Step 2: Assign variables involved in the second clause, $(x^{F} \vee z^{T})$, onto the memory complexes obtained in step 1; selectively collect the complexes, containing x^{F} or z^{T}. At this stage, the collected memory complexes satisfy the first two clauses.

Step 3: Assign variables involved in the third clause, $(y^{F} \vee z^{F})$, onto the memory complexes generated in step 2; selectively collect those complexes, containing y^{F} or z^{F}. At this stage, the collected memory complexes satisfy all the three clauses in the formula.

Step 4: Readout. Assign the undefined variables onto memory complexes obtained in step 3 and read the (x, y, z) assignment(s) in the final pool.

3.3. Implementation of step 1

To implement step 1 that deals with the clause $(x^{T} \vee y^{F})$, the initial memory strands, say in *pool 0*, are split into two portions, as illustrated in Fig. 3(a). We add stickers x^{T} to the first portion of memory strands. The resulted memory complexes then carry a uniform binary code $(x^{T}, ?, ?)$, where the "?" signs indicate that the variables y and z are not defined, because none of stickers y^{T}, y^{F}, z^{T} and z^{F} are added. This "?" notation is used for each undefined variable in the rest of description. Note in Fig. 3, for more clarity in illustration, we use sequence notation $x^{T}x^{F}y^{T}y^{F}z^{T}z^{F}$, instead of binary codes, to describe the formation of memory complexes. Meanwhile, each sequence notation matches a binary code. For instance, $x^{T}x^{F}y^{T}y^{F}\boxed{z^{T}}\,z^{F}$ matches $(?, ?, z^{T})$, in which the shaded component represents an annealed site and the variable is defined by that annealed sticker. Sickers y^{F} are added to the second portion to form memory complexes $(?, y^{F}, ?)$ where the values of x or z are not assigned. Because the memory complexes $(x^{T}, ?, ?)$ and $(?, y^{F}, ?)$ satisfy $x = x^{T}$ or $y = y^{F}$, any further assignment for the undefined variables in the memory complexes will not conflict with the clause $(x^{T} \vee y^{F})$. For example, assigning $y = y^{T}$ onto $(x^{T}, ?, ?)$ and $(?, y^{F}, ?)$ leads to the formation of $(x^{T}, y^{T}, ?)$ and $(?, y^{F}, ?)$, where the satisfaction to $(x^{T} \vee y^{F})$ still holds. Besides, the undefined variables are open to meet conditions implicit in the remaining clauses. These two portions of complexes are mixed and poured into columns that probe the existence of sticker x^{T} or y^{F} on the memory complexes. This separation process is to exclude the memory strands without annealing sticker x^T or y^F from the updated pool. The collected complexes form the *pool 1* and they satisfy the first clause ($x^T \vee y^F$).

3.4. Implementation of separation process

The above-mentioned separation process is achieved with oligonucleotide probes immobilized in columns. The x^T (or y^T or z^T) probe is a 10 base long single-stranded sequence and identical to the first 10 bases in sticker x^T (or y^T or z^T), while the x^F (or y^F or z^F) probe is a 10 bases long strand and identical to the last 10 bases in sticker x^F (or y^F or z^F). To filter out the memory strands without sticker x^T or y^F, the mixture of memory complexes is poured into a column with x^T probe first. This allows memory complexes with sticker x^T to pass through the column. While those complexes without sticker x^T will be captured in the column, because the first 10 bases in the *x*-variable subsection of memory complexes are not covered by a true value sticker, x^T, and are available to bind to the probe. Both the pass-through part and the retained part are collected separately, but the retained part is poured into a second column with y^F probe. Again, the complexes with sticker y^F will pass though. The x^T and y^F related pass-though parts are collected as *pool 1*.

3.5. Implementation of step 2

Step 2 deals with the clause ($x^F \vee z^T$). We split the *pool 1* into two portions, each portion contains both complexes (x^T, ?, ?) and (?, y^F, ?). This process assumes that the mix operation is performed perfectly. As shown in Fig. 3(b), step 2 is achieved by adding stickers x^F in one portion to form memory complexes (x^T, ?, ?) and (x^F, y^F, ?), and adding stickers z^T to the second portion to form memory complexes (x^T, ?, z^T) and (?, y^F, z^T). Note in the first portion the memory complexes with the value $x = x^T$ inherited from step 1 are not replaced by the currently added stickers x^F, because of the 10-base overlapping sequence between strands x^T and x^F. Among the newly updated memory complexes,

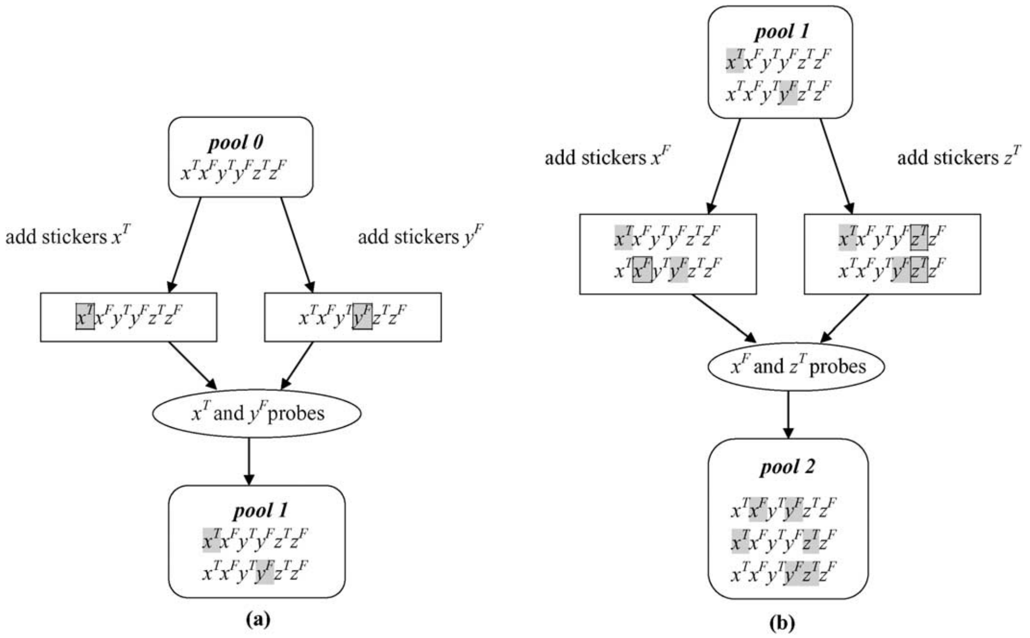
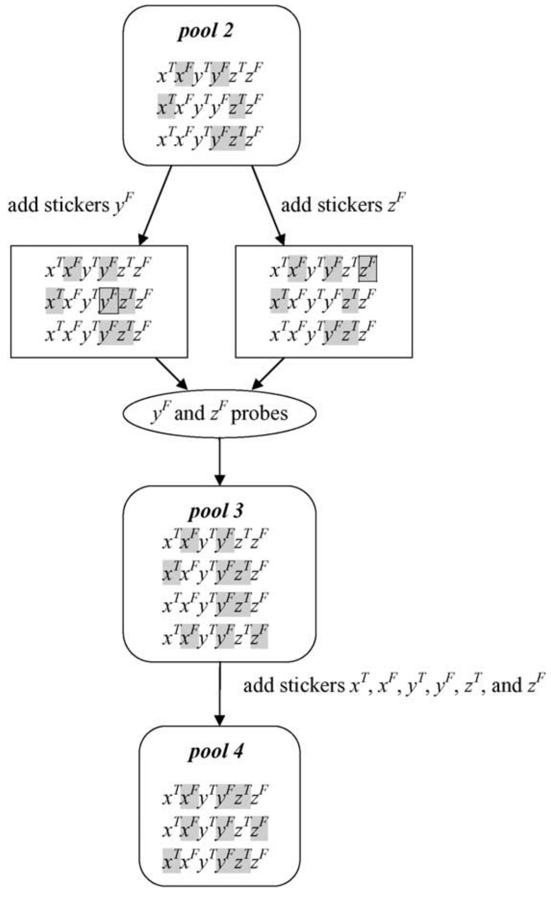


Fig. 3. Steps to assign values to the memory strands. (a) step 1, (b) step 2 and (c) step 3. Note that each shaded value represents a corresponding variable annealed by a sticker, and thus, the variable is defined by the annealed sticker value; otherwise the values of variables are undefined. Moreover, a boxed shaded variable correlates to a sticker annealed in the current step, whereas the unboxed shaded ones are annealed in earlier steps.



(c)

Fig. 3. (*Continued*.)

(x^{T}, ?, ?), (x^{F}, y^{F}, ?), (x^{T}, ?, z^{T}) and (?, y^{F}, z^{T}), we keep those with stickers x^{F} or z^{T} for they satisfy the second clause ($x^{\mathrm{F}} \vee z^{\mathrm{T}}$). This separation process is done with x^{F} and z^{T} probes. The memory complexes (x^{F}, y^{F}, ?), (x^{T}, ?, z^{T}) and (?, y^{F}, z^{T}) survive from the separation process and satisfy the condition $(x^{\mathrm{T}} \vee y^{\mathrm{F}}) \wedge (x^{\mathrm{F}} \vee z^{\mathrm{T}})$. Whereas ($x^{\mathrm{T}}$, ?, ?) is eliminated because it satisfies only the $(x^{\mathrm{T}} \vee y^{\mathrm{F}})$ but fails to satisfy $(x^{\mathrm{T}} \vee y^{\mathrm{F}}) \wedge (x^{\mathrm{F}} \vee z^{\mathrm{T}})$. These qualified memory complexes form *pool 2*.

The separation process in step 2 is achieved with probes x^{F} and z^{T}. Among the newly updated memory complexes, (x^{T}, ?, ?), (x^{F}, y^{F}, ?), (x^{T}, ?, z^{T}) and (?, y^{F}, z^{T}), the passage of probe x^{F} will retain (x^{T}, ?, ?), (x^{T}, ?, z^{T}) and (?, y^{F}, z^{T}), whereas (x^{F}, y^{F}, ?) will pass through and be collected. Further we collect those retained in the x^{F}-column and pour them in to a second column with probe z^{T}. This time (x^{T}, ?, z^{T}) and (?, y^{F}, z^{T}) will pass through and (x^{T}, ?, ?) will be captured again in the column for the lack of sticker z^{T}. The complexes (x^{F}, y^{F}, ?), (x^{T}, ?, z^{T}), and (?, y^{F}, z^{T}) form the *pool 2*. Note that as we change the selection process by applying probes z^{T} first and then x^{F}, the result will be the same.

3.6. Implementation of step 3

Satisfaction of ($y^{\mathrm{F}} \vee z^{\mathrm{F}}$) is considered in step 3. The *pool 2* is split into two portions: one for addition of stickers y^{F} and the other, for z^{F}. As indicated in Fig. 3(c), the portion with addition of stickers y^{F} are updated to (x^{F}, y^{F}, ?), (x^{T}, y^{F}, z^{T}), and (?, y^{F}, z^{T}); the portion with addition of stickers z^{F} are updated to (x^{F}, y^{F}, z^{F}), (x^{T}, ?, z^{T}), and (?, y^{F}, z^{T}). Again, we keep those memory complexes satisfying the third clause ($y^{\mathrm{F}} \vee z^{\mathrm{F}}$), i.e., those containing y^{F} and/or z^{F} stickers, by applying y^{F} and z^{F} probes. The surviving memory complexes, mixed to from *pool 3*, are (x^{T}, y^{F}, z^{T}), (x^{F}, y^{F}, z^{F}), (x^{F}, y^{F}, ?), and (?, y^{F}, z^{T}). They satisfy the formula $F = (x^{\mathrm{T}} \vee y^{\mathrm{F}}) \wedge (x^{\mathrm{F}} \vee z^{\mathrm{T}}) \wedge (y^{\mathrm{F}} \vee z^{\mathrm{F}})$. For complexes with undefined variables, such as (x^{F}, y^{F}, ?) and (?, y^{F}, z^{T}), they satisfy the studied Boolean formula with the already defined variables and do not care what the undefined variables are. Therefore, we add all the sticker molecules x^{T}, x^{F}, y^{T}, y^{F}, z^{T} or z^{F} into *pool 3* to assign the undefined variables. Consequently, complexes (x^{F}, y^{F}, ?) can randomly pick either stickers z^{T} or z^{F} and convert into (x^{F}, y^{F}, z^{F}) and (x^{F}, y^{F}, z^{T}), whereas (?, y^{F}, z^{T}) becomes (x^{T}, y^{F}, z^{T}) and (x^{F}, y^{F}, z^{T}), based on the same rationale. Meanwhile, the addition of stickers will not affect the memory complexes (x^{T}, y^{F}, z^{T}) and (x^{F}, y^{F}, z^{F}), since these complexes do not contain available sites for additional sticker molecules. The assignments (x^{F}, y^{F}, z^{F}), (x^{T}, y^{F}, z^{T}), and (x^{F}, y^{F}, z^{T}) are the solutions of the studied formula. These qualified complexes are collected to form *pool 4* for readout process.

The separation process in step 3 is achieved with probes y^{F} and z^{F}. Among the newly updated memory complexes, (x^{F}, y^{F}, ?), (x^{T}, y^{F}, z^{T}), (?, y^{F}, z^{T}), (x^{F}, y^{F}, z^{F}), and (x^{T}, ?, z^{T}), the passage of probe y^{F} will retain (x^{T}, ?, z^{T}), whereas (x^{F}, y^{F}, ?), (x^{T}, y^{F}, z^{T}), (?, y^{F}, z^{T}), and (x^{F}, y^{F}, z^{F}) will pass through and be collected. Further we collect those retained in the y^{F} column and pour them into a second column with probe z^{F}. Complexes

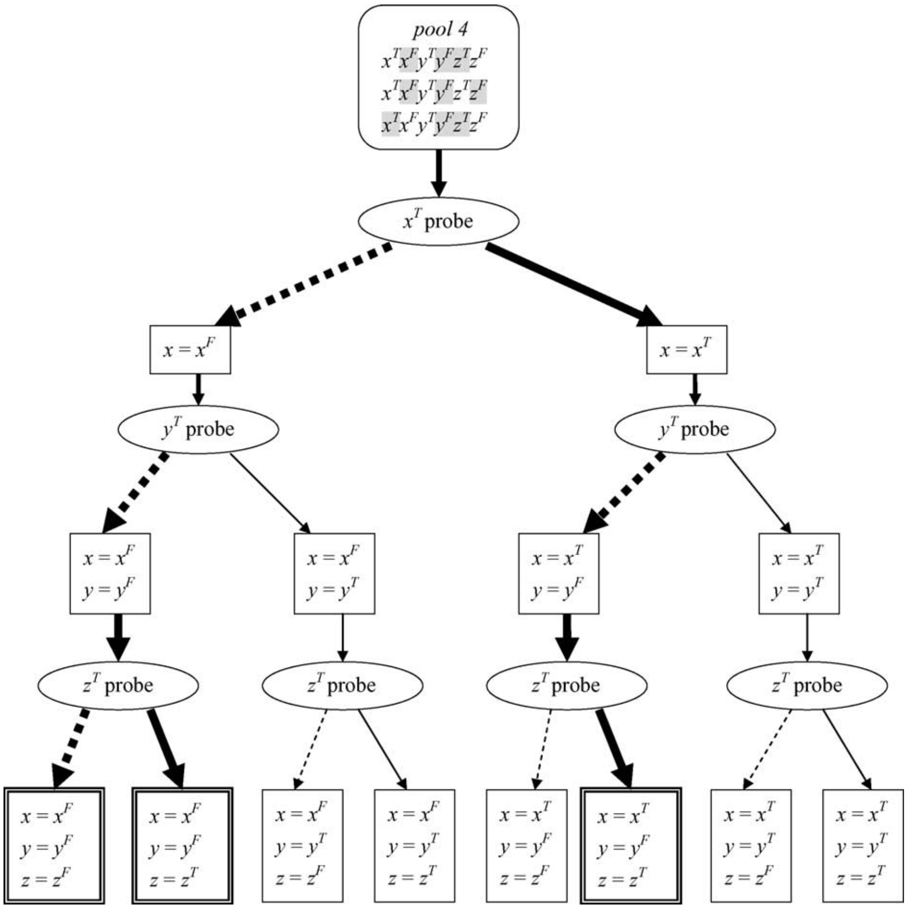

pool 4
$x^T x^F y^T y^F z^T z^F$
$x^T x^F y^T y^F z^T z^F$
$x^T x^F y^T y^F z^T z^F$
x^T probe
$x = x^F$
$x = x^T$
y^T probe
y^T probe
$x = x^F$ $y = y^F$
$x = x^F$ $y = y^T$
$x = x^T$ $y = y^F$
$x = x^T$ $y = y^T$
z^T probe
$x = x^F$ $y = y^F$ $z = z^F$
$x = x^F$ $y = y^F$ $z = z^T$
$x = x^F$ $y = y^T$ $z = z^F$
$x = x^F$ $y = y^T$ $z = z^T$
$x = x^T$ $y = y^F$ $z = z^F$
$x = x^T$ $y = y^F$ $z = z^T$
$x = x^T$ $y = y^T$ $z = z^F$
$x = x^T$ $y = y^T$ $z = z^T$

Fig. 4. The readout process. The bold solid lined paths and bold dashed lined paths are the through paths, otherwise they are the paths not through in the proposed experiment. A dashed lined route means that the memory complexes do not contain the sticker matching the probe; a solid lined route is for memory complexes with stickers matching the probe. Each path leads to a box of assignment and the double-lined boxes are marked for the through paths. The assignments in double-lined boxes are solutions to the studied Boolean formula.

(x^T, ?, z^T) still get retained in the second column and nothing will pass through. Complexes (x^F, y^F, ?), (x^T, y^F, z^T), (?, y^F, z^T), and (x^F, y^F, z^F) form the *pool 3*.

3.7. Implementation of readout process in step 4

The readout process is designed to tell the contents of *pool 4*. This again utilizes the x^T, y^T, and z^T probes. As shown in Fig. 4, the *pool 4* is poured into a column that probes the existence of sticker x^T on the memory complexes. This separates the *pool 4* into two portions, one with $x = x^T$ and the other with $x = x^F$. We name such a process *x*-separation. Further, for each of these two newly separated portions we collect them and apply y^T probes for *y*-separation, respectively. This again allows each tube obtained from *x*-separation to be divided into two portions, one detected with $y = y^T$ and one detected with $y = y^F$. Likewise, the collected memory complexes in different tubes are, respectively, poured into columns with z^T probes for *z*-separation. In final collection of memory complexes in different tubes, each tube represents a three-digit binary code of (x, y, z) assignment on the memory complexes. Fig. 4 shows the detailed flow chart of readout process. Note in the studied case, some of the paths in Fig. 4 are not through in the proposed experiment, because some tubes may collect nothing, and thus, are not available for further separation processes. In Fig. 4, we demonstrate all possible paths and mark the through paths with bold lines. As the contents in the *pool 4* are known as (x^F, y^F, z^F), (x^T, y^F, z^T) and (x^F, y^F, z^T), the through paths lead to the double-lined boxes for the corresponding assignments.

4. Summary of algorithm

Let's now turn to the general case. Any SAT problem involving k variables on m clauses, with n_i literals in the ith clause ($1 \leqq i \leqq m$), can be solved with m steps plus one final readout process with our algorithm. The variety of molecules involved is $2k$ sticker strands and one memory strand with k subsections. Computing begins with dividing a tube of memory strands into n_1 tubes for adding n_1 sorts of stickers, one tube for one sort of sticker that corresponds to a value involved in the first clause. The n_1 tubes, each of them contains one of the n_1 sorts of resulted memory complexes, are poured together to pass n_1 sorts of probes. The collected complexes form *pool 1*, which is an ensemble of OR situations among the n_1 literals in the first clause. Further step is repeatedly done by dividing the *pool* $i-1$ into n_i tubes for adding n_i sorts of stickers, mixing the resulted n_i tubes' contents, and pouring the mixture through n_i probes for the appearance of the n_i stickers related to the ith clause. Memory complexes match at least one of the probes are collected to form *pool i*. Such a separation process is to filter out those memory complexes that do not contain any value asked in the ith clause. The filter-out complexes result from the fact that their n_i literals in the ith clause are all already defined, in the preceding $i-1$ step(s), oppositely to the values required in the ith clause.

For an unsolvable Boolean formula, say, the jth clause conflicts with conditions in the first $j-1$ clauses, the passages of n_j sorts of probes will collect no qualified sequences and lead the computing to stop. This termination is due to all of the considered n_j variables are already defined as the opposite values in the previous $j-1$ steps, and consequently, the memory complexes in *pool* $j-1$ can not take any value required in the jth clause.

Right before the readout process, we assign both true and false values to each undefined variable by adding all sorts of stickers to the final pool, *pool m*. An unannealed variable site randomly takes either a true or false sticker, for both assignments will not affect the satisfiability provided by other variables.

5. Concluding remark

The major advantage of DNA computing is its high parallelism in data processing which leads most DNA computing procedures to be blueprinted as a construction of a variety of possible solutions followed by a brute force search for the answer sequences. However, such an approach imposes limitations on the problem size. That is, as the number of variables increases dramatically, the initial data pool becomes too huge to handle experimentally. To bypass the drawback induced by the size problem, the memory strands in our model are designed to take information through steps and eventually encode various combinations of information to represent the answer sequences. That is, the selection process is implemented in data pool construction and the size of data pool is always under control. With this algorithm, though the proposed experiment is very ideal and ignores many complicate factors, such as temperature control for sticker annealing, purity of separation process, and amplification of answer sequences for readout process, we anticipate to bring a new fashion on DNA computing. Meanwhile, based on the same architecture of computing algorithm, we currently adopt DNA array to our experimental version of SAT solution, which also does not take brute force search strategy. The design and result will be presented in latter publication.

Acknowledgement

This work is supported by National Science Council in Taiwan (NSC 93-2113-M-214-001).

References

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

Braich, R.S., Chelyapov, N., Johnson, C., Rothemund, P.W.K., Adleman, L., 2002. Solution of a 20-variable 3-SAT problem on a DNA computer. Science 296, 499–502.

Chen, K., Ramachandran, V., 2001. In: Condon, A., Rozenberg, G. (Eds.), DNA Computing. Springer, pp. 199–208.

Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C., 2001. Introduction to Algorithms, second ed. MIT Press, Boston, MA.

Díaz, S., Esteban, J.L., Ogihara, M., 2001. In: Condon, A., Rozenberg, G. (Eds.), DNA Computing. Springer, pp. 209–219.

Faullhammer, D., Cukras, A.R., Lipton, R.J., Landweber, L.F., 2000. Molecular computation: RNA solutions to chess problems. Proc. Natl. Acad. Sci. U.S.A. 97, 1385–1389.

Gillmor, S.D., Rugheimer, P.P., Lagally, M.G., 2002. Computing with DNA on surfaces. Surf. Sci. 500, 699–721.

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

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

Liu, Q., Wang, L., Frutos, A.G., Condon, A.E., Corn, R.M., Smith, L.M., 2000. DNA computing on surfaces. Nature 403, 175–179.

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

Păun, G., Rozenberg, G., Salomaa, A., 1998. DNA Computing: New Computing Paradigms. Springer-Verlag, Berlin, pp. 117–149.

Roy, B., Tlusty, T., Libchaber, A., 2002. Protein–DNA computation by stochastic assembly cascade. Proc. Natl. Acad. Sci. U.S.A. 99 (18), 11589–11592.

Sakamoto, K., Gouzu, H., Komiya, K., Kiga, D., Yokoyama, S., Yokomori, T., Hagiya, M., 2000. Molecular computation by DNA hairpin formation. Science 288, 1223–1226.

Yoshida, H., Suyama, A., 1999. DNA based computers 5. In: Winfree, E., Gifford, D.K. (Eds.), DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 54. American Mathematical Society, Providence, RI, pp. 9–20.