DNA computing by blocking

Rozenberg, G.; Spaink, H.P.

ELSEVIER

Theoretical Computer Science 292 (2003) 653–665

Theoretical Computer Science

www.elsevier.com/locate/tcs

DNA computing by blocking

G. Rozenberg[a,b,*], H. Spaink[c]

[a] *Leiden Institute of Advanced Computer Science (LIACS) and Leiden Center for Natural Computing (LCNC), Leiden University, P.O. Box 9512, Niels Bohrweg 1, 2300 RA Leiden, Netherlands*
[b] *Department of Computer Science, University of Colorado at Boulder, Boulder, CO 80309, USA*
[c] *Clusius Laboratory, Institute of Molecular Plant Sciences (IMP) and Leiden Center for Natural Computing (LCNC), Leiden University, Wassenaarseweg 64, 2333 AL Leiden, Netherlands*

Received December 2000; received in revised form February 2001; accepted March 2001
Communicated by A. Salomaa

Abstract

We present a method for molecular computing which relies on blocking (inactivating) this part of the total library of molecules that does not contribute to (finding) a solution—this happens essentially in one biostep (*after* the input has been read). The method is explained by presenting a DNA based algorithm for solving (albeit in the theoretical sense only!) the satisfiability problem.

Keywords: Molecular computing; DNA computing; Filtering methods; Satisfiability problem; PCR

1. Introduction

DNA computing (or, more generally, molecular computing) is a modern and very active research area investigating the use of biomolecules for the purpose of computing (see, e.g., [1, 3, 9]). It is a genuinely interdisciplinary research area, where molecular biologists and computer scientists cooperate to achieve an exciting goal: to develop biomolecular computers which in the future may replace, or complement, silicon based computers.

Roughly speaking, one can distinguish two major lines of research in DNA computing: the theoretical line concerned with models, algorithms and paradigms for DNA

* Corresponding author. Leiden Institute of Advanced Computer Science (LIACS) and Leiden Center for Natural Computing (LCNC), Leiden University, P.O. Box 9512, Niels Bohrweg 1, 2300 RA Leiden, Netherlands. Tel.: +31-71-277-063; fax: +31-71-276-985.
E-mail address: rozenber@wi.leidenuniv.nl (G. Rozenberg).

0304-3975/03/$ - see front matter
PII: S0304-3975(01)00194-3

computing, and the experimental line concerned with the design of laboratory experiments testing the biochemical feasibility of theoretical ideas.

A lot of theoretical research considers some basic properties of biomolecules and some basic biomolecular operations, formulates (the essence of) them as formal properties and operations, and then investigates models, algorithms, and paradigms based on these formal properties and operations. Our paper falls into this line of research: we use the basic properties of DNA molecules (such as the Watson–Crick complementarity, see, e.g., [2, 9]), and the basic laboratory operations (such as the polymerase chain reaction, see, e.g., [2, 9]) to formulate the paradigm of blocking and the algorithms based on it. In doing this, we are not concerned here with either the biochemical feasibility of our algorithms or their performance (such as the time or the volume complexity). In this sense this paper is *purely theoretical*. However, we address many "practical issues" in the companion paper [11].

We have taken an effort to write this paper in the interdisciplinary spirit, so that it is accessible to both researchers in (theoretical) computer science and researchers in molecular biology—of course providing that they are sufficiently interested in (learning about) DNA computing.

2. DNA molecules: structure, notation and basic operations

DNA molecules are polymers that are built from "simple" monomers called *nucleotides* (DNA stands for "deoxyribonucleic acid"). Each nucleotide consists of three basic components: *sugar, phosphate*, and *base*. The sugar molecule has five attachment points (carbon atoms) which are labeled $1'$ through $5'$: the phosphate molecule is attached to $5'$, and the base is attached to $1'$. There are four possible bases, denoted by A, C, G, and T. Since nucleotides may differ only in their bases, we identify nucleotides with their bases—thus there are only four types of nucleotides, also denoted by A, C, G, and T.

Nucleotides can form *single stranded DNA molecules*: two consecutive nucleotides in such a strand bind through a strong (*covalent*) bond between the (hydroxyl group on the) $3'$-attachment point of one nucleotide and the (phosphate group on the) $5'$-attachment point of the other nucleotide; this bond is called the *phosphodiester bond*. In the so formed single stranded DNA molecule one of its ends (nucleotides) has the $5'$-attachment point available for binding with yet another nucleotide, while the other end has the $3'$-attachment point available for binding. Since chemically the $5'$-attachment point can be easily distinguished from the $3'$-attachment point, a single stranded DNA molecule has easy to establish *polarity* (*orientation*): its nucleotide sequence can be read either from its $5'$-end to its $3'$-end, or the other way around (it turns out that $5'$–$3'$ polarity is favoured by nature: the biological information is encoded in the $5'$–$3'$ orientation).

The basic feature of the four bases $\{A, C, G, T\}$ is the pairwise affinity, A with T, and C with G: we say that bases A and T are *complementary*, and so are C and G. This

complementarity (called the *Watson–Crick complementarity*) underlies the formation of double stranded DNA molecules from the single stranded ones. Two single stranded DNA molecules can bind elementwise through their bases forming weak (*hydrogen*) bonds. These bonds can form only between complementary bases, hence between A and T, and between C and G. Moreover, two single stranded DNA molecules can form a double stranded DNA molecule only if the two strands are of opposite orientation (meaning that the first nucleotide at the 5′-end of one strand binds to the first nucleotide at the 3′-end of the other strand; the second nucleotide from the 5′-end of the first strand binds to the second nucleotide from the 3′-end of the second strand, etc.).

Since each nucleotide can be identified by one of the four letters from the alphabet $\mathcal{N} = \{A, C, G, T\}$, a single stranded DNA molecule can be denoted by a string over $\mathcal{N}$, where the left-to-right orientation of a string corresponds to the 5′–3′ polarity of the strand. Thus, e.g., *TCTAG* denotes the single stranded DNA molecule with a T-nucleotide at its 5′-end, a C-nucleotide bonded to it, then another T-nucleotide bonded to this C-nucleotide, etc. This single stranded DNA molecule could also be denoted by the more explicit notation 5′–*TCTAG*–3′.

To denote double stranded DNA molecules we will use "double strings". Thus, e.g.,

5′–*CGAATG*–3′
3′–*GCTTAC*–5′

denotes the double stranded DNA molecule with one of the single strands being *CGAATG* and the other one being *CATTCG* (both read in the 5′–3′ direction). The usual convention is that the upper string (read left-to-right) represents one of the strands in the 5′–3′ orientation, while the lower string (read left-to-right) represents the other strand in the 3′-5′ orientation. Using this convention, the double stranded DNA molecule given above is represented by the double string

CGAATG
GCTTAC

as well as by the double string

CATTCG
GTAAGC.

The double string

TCGT
ACAGCA

represents a double stranded DNA molecule that is not a perfect duplex, i.e., some nucleotides in one strand do not have the corresponding complementary nucleotides in the other strand.

We need to stress here that our presentation of the basic structure of DNA molecules is very simplified, although adequate for the purpose of this paper. In particular, the reader should be aware that double stranded molecules form a quite sophisticated spatial structure—the famous double helix discovered by Watson and Crick [12].

One can separate the two strands of a double stranded DNA molecule by, e.g., heating the solution containing the molecule. Since the hydrogen bonds between the two strands are much weaker than the phosphodiester bonds between the consecutive nucleotides within a single strand, such a separation will not break the single strands. By cooling down the solution, the separated strands will fuse together forming the original double strand.

The fusing process is called *annealing* or *hybridization*, and the separation process is called *melting* or *denaturation*. Once again we stress that our presentation of basic molecular biology is very simplified. Thus, e.g., the cooling down for the purpose of annealing must proceed slowly enough, so that the corresponding complementary bases have enough time to find each other.

3. Polymerase chain reaction

In genetic engineering one is often faced with a situation when one has to detect whether a specific molecule is present in "the sea of other molecules" (e.g., among a trillion of other molecules!). Fortunately, a technique for doing just that has been devised in 1985 (and it has really revolutionized molecular biology). It is called *polymerase chain reaction* (PCR), and it is based upon the activity of the enzyme called DNA *polymerase* (see, e.g., [2, 9]). This enzyme turns a single stranded DNA molecule into a double stranded one by simply adding to each nucleotide in the single strand its complementary nucleotide in the other strand (and forming the phosphodiester bond with the previous nucleotide in this strand). However, polymerase can do this only if there is already a short stretch of nucleotides complementing the $3'$-end of the single stranded molecule. This short stretch (called *primer*) will be then extended, nucleotide by nucleotide, in the $5'$–$3'$ direction until the whole single strand is extended to a double stranded molecule. An example of such an extension is given in Fig. 1, where the primer is $5'$–*CACATTGAC*.

Let α be the double stranded molecule that we are after. In order to apply PCR to search for α, we need to know the two $3'$-borders of α (i.e., short stretches at the $3'$-ends of the two single stranded molecules of α), say β and γ. Let Z_0 be the initial solution (which we want to search for the presence of α), enriched by primers (complementary to β and γ), polymerase and a sufficient supply of nucleotides. The PCR proceeds by repeating the basic cycle consisting of three steps: *denaturation*, *priming* and *extension*, beginning with Z_0. During the denaturation step, the solution is heated so that the hydrogen bonds are destroyed, and hence the double stranded molecules separate into the single stranded molecules (α separates into α_1 and α_2 complementary to each other). Then, during the priming step the solution is cooled

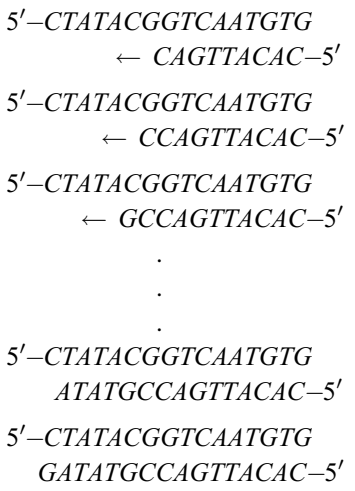

Fig. 1. A primer extension process.

down (typically to about 55°C) so that the primers anneal to the complementary borders of α_1 and α_2 (forming primed molecules α_1' and α_2'). These molecules (α_1' and α_2') are extended to the double stranded molecules during the extension step when the temperature of the solution is increased (typically to 72°C), so that polymerase can extend each of α_1' and α_2' to a molecule identical to α. Consequently, during one basic cycle (denaturation + priming + extension) two copies of α are made, and so, after n repetitions of the basic cycle, *in principle* 2^n copies of α are made (hence PCR is a *very* efficient "copy machine" for DNA molecules).

Thus, PCR dramatically increases the density of α in the solution (if it is there), and this makes the detection of α much easier. It should be noted that during the denaturation step the solution is heated to a temperature close to boiling, and so the polymerase (present in the solution all the time) must survive this high temperature. Fortunately, polymerases isolated from thermophilic bacteria living in thermal springs can do that.

Although in the above description of PCR we have assumed that the sought after molecule is double stranded, the PCR will work also for a single stranded α which will be made into a double stranded molecule in the extension step of the first cycle.

4. Satisfiability

In this section we briefly recall the satisfiability problem for Boolean expressions, see, e.g., [5]. Then, in Section 6 we will present a "molecular algorithm" for solving it, albeit only in theoretical sense. We have chosen to work on this problem, because it is well known and it plays a central role in the family of computational problems (see, e.g., [5]), and because it is very well suited for the use of blocking.

Let $V=\{p_1,\ldots,p_n\}$ be a set of *Boolean variables*—their values may be only 0 and 1 (0 stands for "false" and 1 stands for "true"). A *literal* is either a variable p_i or its *negation* $\neg p_i$, and we say that $p_i, \neg p_i$ are *literals for* p_i.

We consider two logical operations: $\vee$ ("or") and $\wedge$ ("and"). A *clause* E is an expression of the form $\ell_1 \vee \cdots \vee \ell_m$, $m \geqslant 1$, where each ℓ_i is a literal; for the purpose of this paper we may assume that for each variable p_i there is at most one literal for p_i in E. A *Boolean formula* (in conjunctive normal form, CNF) is an expression of the form $E_1 \wedge \cdots \wedge E_t$ where each E_i is a clause.

An *assignment* is a function φ on V which for each p_i has the value either 0 or 1. To compute the value of a literal (for a given assignment φ) we use the rule: $\neg 0 = 1$ and $\neg 1 = 0$. To compute the value of a clause we use the rule: $0 \vee 0 = 0$ and $0 \vee 1 = 1 \vee 0 = 1 \vee 1 = 1$. To compute the value of a formula we use the rule: $0 \wedge 0 = 0 \wedge 1 = 1 \wedge 0 = 0$ and $1 \wedge 1 = 1$.

We say that an assignment φ *satisfies* a formula Φ if the value $\varphi(\Phi)$ of Φ under φ is 1; otherwise φ *falsifies* Φ (and φ is a *falsifier* of Φ). Note that φ falsifies Φ if and only if φ falsifies at least one clause of Φ. We say that Φ is *satisfiable* if there is an assignment satisfying Φ.

Example 4.1. Let $V=\{p_1, p_2, p_3\}$ be a set of variables, and let $\Phi = E_1 \wedge E_2 \wedge E_3$ be a Boolean formula over V such that $E_1 = p_1 \vee \neg p_2$, $E_2 = p_1 \vee p_2 \vee \neg p_3$, and $E_3 = \neg p_1 \vee \neg p_3$.

Let φ_1 be the following assignment: $p_1 = 0$, $p_2 = 1$, $p_3 = 0$. Then $\varphi_1(E_1) = 0 \vee 0 = 0$, $\varphi_1(E_2) = 0 \vee 1 \vee 1 = 1$ and $\varphi_1(E_3) = 1 \vee 1 = 1$. Thus $\varphi_1(\Phi) = 0 \wedge 1 \wedge 1 = 0$. Let φ_2 be the assignment: $p_1 = 0$, $p_2 = 0$, $p_3 = 0$. Then $\varphi_2(E_1) = 0 \vee 1 = 1$, $\varphi_2(E_2) = 0 \vee 0 \vee 1 = 1$, and $\varphi_2(E_3) = 1 \vee 1 = 1$. Thus $\varphi_2(\Phi) = 1 \wedge 1 \wedge 1 = 1$. Hence φ_1 falsifies Φ, φ_2 satisfies Φ, and Φ is satisfiable.

The *Satisfiability Problem* (SAT) is to determine whether or not an arbitrary Boolean formula Φ is satisfiable.

In the sequel, we assume that we have an infinite sequence of variables $p_1, p_2, p_3, \ldots$ and whenever we consider the case of n variables, the variables are $p_1, p_2, \ldots, p_n$.

5. Blockers

In this section we introduce the basic idea of blocking. It is based on the Watson–Crick complementarity, and the specific form of blocking is dependent on the way that the assignments of variables are coded by DNA molecules.

To start with, we need to code for each (Boolean) variable p_i its two possible values $p_i = 1$ and $p_i = 0$. Let $q_i^{(1)}$ be a sequence of nucleotides in a single strand coding the value $p_i = 1$, and let $q_i^{(0)}$ be a sequence of nucleotides in a single strand coding the value $p_i = 0$.

Example 5.1. The coding $q_i^{(1)} = A$ and $q_i^{(0)} = C$ for all $1 \leqslant i \leqslant n$ is independent of a variable. The value 1 is always coded by A and the value 0 is always coded by C.

Now, we can code each assignment of variables by a single strand. To this aim, for a given number of variables n, an *n-strand* is a strand of the form $f_1 f_2, \ldots, f_n$ where each f_i is either $q_i^{(1)}$ or $q_i^{(0)}$.

The set of all n-strands is denoted by S_n; thus S_n represents (codes) all possible assignments.

Example 5.2. For $n = 3$ and the coding of values of single variables as in Example 5.1, AAC is a 3-strand coding the assignment φ such that $\varphi(p_1) = \varphi(p_2) = 1$ and $\varphi(p_3) = 0$.

For an n-strand s, we use $asg(s)$ to denote the corresponding assignment φ, and for an assignment φ we use $str(\varphi)$ to denote the corresponding n-strand.

A *blocker* of an n-strand s is its complement, denoted by $b(s)$. Now, given a Boolean formula $\Phi = E_1 \wedge E_2 \wedge \cdots \wedge E_m$ over n variables, for each clause E_i a *blocker* of E_i is a blocker of an n-strand s such that $asg(s)$ is a falsifier of E_i. The set of all blockers for E_i is denoted by $B(E_i)$. Then the set of all blockers for Φ, denoted by $B(\Phi)$, is the union of $B(E_i)$ for all clauses E_i of Φ; thus $B(\Phi) = B(E_1) \cup \cdots \cup B(E_m)$.

Example 5.3. Let $n = 3$ and let $\Phi = E_1 \wedge E_2$, where $E_1 = \neg p_1 \vee p_3$ and $E_2 = \neg p_1 \vee \neg p_2 \vee \neg p_3$. The falsifiers for E_1 are φ_1 and φ_2, where $\varphi_1(p_1) = 1$, $\varphi_1(p_2) = \varphi_1(p_3) = 0$, and $\varphi_2(p_1) = \varphi_2(p_2) = 1, \varphi_2(p_3) = 0$. The falsifier for E_2 is φ_3 such that $\varphi_3(p_1) = \varphi_3(p_2) = \varphi_3(p_3) = 1$.

Hence if we use the coding from Example 5.1, then the blockers for E_1 are GGT and GTT (recall that, unless indicated otherwise, we write DNA sequences in the 5′–3′ direction), because $str(\varphi_1) = ACC$ and $str(\varphi_2) = AAC$. The blocker for E_2 is TTT because $str(\varphi_3) = AAA$. Hence the set of blockers for Φ is $B(\Phi) = \{GGT, GTT, TTT\}$.

The molecular computing method that we will present in Sections 6 and 7 consists of blocking a part of the library of molecules (S_n), and then amplifying the remaining part (if any) using PCR. Thus our representation of n-strands and blockers is very simplified. Since one needs to prime strands to be amplified by PCR, all the n-strands will have special prefixes and suffixes that are needed for a PCR. The blockers are then modified accordingly.

Recall that, PCR consists of iterating the basic cycle, which begins with denaturation that unblocks the initially blocked molecules. These molecules can be then primed, and copied during the iteration process—*and this should not be done* (we do not want blocked assignments to be multiplied). This problem can be solved by using PNA rather than DNA molecules for blocking. PNA molecules (see, e.g., [10]), are single stranded chemically synthesized DNA analogues which anneal sequence—selectively to complementary DNA molecules (PNA stands for "peptide nucleic acid"). However, PNA molecules anneal to the complementary DNA molecules with higher affinity than

the corresponding DNA molecules. This means that if we use PNA rather than DNA blockers, then when the PCR solution is cooled down after the denaturation (to facilitate priming) the PNA blockers will anneal to their complementary DNA molecules *before* (hence at a higher temperature than) DNA primers can anneal (to the complementary 3′-ends of DNA molecules). This means that PNA molecules win this "annealing competion" with primers, and consequently the molecules blocked by PNA cannot be primed. Thus, the problem of preventing the blocked molecules to be also multiplied by PCR can be solved in this way.

As stated already, our presentation of basic notions and techniques of molecular biology is very simplified, and in our algorithms we use formal abstractions of these simplified concepts. Now, for a given library S_n of all n-strands, if we add a set of PNA blockers B to (a solution containing) S_n, then all the n-strands which are complementary to the strands in B become "inactivated", and one can proceed then to process the remaining strands, e.g., by PCR. This is the main idea behind the use of blocking for solving problems such as SAT.

6. An algorithm for SAT

We are ready now to present a molecular algorithm based on the blocking principle for solving SAT (albeit in the theoretical sense only!). Thus, assume that we are given a Boolean formula Φ over n variables.

We begin with an initial solution Z_0 that contains the set S_n of all n-strands. To know S_n, we need to know only the number of variables n (without knowing Φ). Thus, we assume that such a solution is prepared in advance—it is a "ready product on a shelf".

Here is an algorithm ($\mathscr{A}_1$) for solving SAT.

Algorithm $\mathscr{A}_1$.

Input: A Boolean formula Φ of n variables.

0. Prepare $B(\Phi)$
1. Add $B(\Phi)$
2. *PCR*
3. *PCR* Successful?
   If so, go to 5
   If not, go to 4.
4. Output "NO" and Stop.
5. Output "YES"
6. Stop

Once we know the input formula Φ, we prepare $B(\Phi)$, and in Step 1 we add it to Z_0 obtaining Z_1. The intention of Step 1 is to "block" (by annealing) all the n-strands which represent assignments that falsify Φ.

In Step 2, Z_1 is PCR'ed and Z_2 is obtained. Here the only n-strands that can be successfully multiplied by PCR are the n-strands that have not been blocked in Step 1—these are the strands that PCR is searching for. But these are precisely the n-strands s such that the assignment $asg(s)$ satisfies Φ. Thus, the PCR here is successful if and only if there exists an assignment satisfying Φ.

In Step 3 we check whether or not the PCR from Step 2 was successful. This is the case if Z_2 contains "clearly more" DNA molecules than Z_1.

If the PCR was not successful, then we proceed to Step 4, print "NO", and stop.

If the PCR was successful, then we proceed to Step 5, print "YES", and stop in Step 6 (the reason for having this stop instruction in a separate step will be clear in Section 7, where $\mathscr{A}_1$ will be extended to the algorithm $\mathscr{A}_2$).

It must be clear by now that this algorithm prints "YES" (and stops) if and only if Φ is satisfiable.

We would like to comment here that using PCR to check the existence of a solution (an unblocked molecule) is a very "elegant" and effective detection method: just by visual inspection when combined with making molecules visible, e.g., by using DNA-specific dyes. The detection of (the existence of) a solution is often a major problem in molecular computing.

Example 6.1. We continue Example 5.3. Here $n=3$ and $S_3=\{CCC, CCA, CAC, CAA, ACC, ACA, AAC, AAA\}$. Since strands from $B(\Phi)$ will anneal to their complements in S_3 (in Z_0), the set of single strands in Z_1, denoted $ss(Z_1)$, equals $S_3-\overline{B(\Phi)}$, where $\overline{B(\Phi)}$ is the set of Watson–Crick complements of molecules in $B(\Phi)$. Hence $ss(Z_1)=\{CCC, CCA, CAC, CAA, ACA\}$. It is easily seen that indeed $asg(ss(Z_1))$ is the set of all assignments satisfying Φ. Since this set is non-empty, the PCR from Step 2 will be successful, and so the algorithm will output "YES" and stop.

We feel that the following comments concerning the above algorithm are needed here, even though the basic laboratory implementation problems are discussed in [11].

(1) We may construct $B(\Phi)$ by reading Φ from left to right, clause by clause, as follows. Let E be a clause of Φ, and we assume that literals in E are ordered according to the order $p_1,\ldots,p_n$ of variables. Let, e.g., $E=p_1\vee p_2\vee\neg p_4$, where $n=4$. Reading E from left to right we can spell out the falsifiers of E: $p_1=0$, $p_2=0$, $p_3=$ "any value", $p_4=1$. Thus if a variable p_i is present in E, then we set $p_i=0$, and if $\neg p_i$ is present in E, then we set $p_i=1$. If neither p_i nor $\neg p_i$ is present in E, then we set "any value" which means that p_i can be either 0 or 1.

The set of blockers of E is then the set of complements of the n-strands that code the falsifiers. Let, e.g., E be as given above, and let the coding be the one given in Example 5.1. Then, reading E (or falsifiers of E) from right to left we can spell out the blockers of E: "first T", "then either G or T", "then G", "then G". In this way we get 2 blockers: $TGGG$ and $TTGG$. Spelling out the blockers while reading E from left to right may be considered as giving instructions (to a "robot") for synthesizing the set of blockers for the clause considered. The reason for reading now E from left

to right is that the chemical synthesis of DNA proceeds in the 3′–5′ direction. Hence for E as above the synthesis would go as follows:

- first G: take a solution R_1 with "enough G" (each G nucleotide is hooked to a solid support at its 3′-end);
- then G: attach G to all the free 5′-ends of molecules in R_1 getting in this way R_2;
- then either G or T: divide R_2 into two solutions $R_{2,1}, R_{2,2}$ of equal volume, attach G to all the free 5′-ends of molecules in $R_{2,1}$ getting $R_{2,1,G}$, attach T to all the free 5′-ends of molecules in $R_{2,2}$ getting $R_{2,2,T}$, then mix $R_{2,1,G}$ with $R_{2,2,T}$ getting R_3;
- then T: attach T to all the free 5′-ends of molecules in R_3 getting in this way R_4—clearly R_4 contains "enough" (and "equal amounts") of all the blockers of E.

If neither p_1 nor $\neg p_1$ is present in a clause, then the initial mixture R_1 will have "enough G" and "enough T" hooked to a solid support, and the synthesis proceeds as outlined above. (Brown [2] is a good reference on the chemical synthesis of DNA strands.)

From a purely theoretical point of view one can also imagine a storage system (library of molecules) such that spelling out a blocker gives the address of the blocker in the storage system. Once the address has been spelled out, the blocker molecules from this storage address ("enough" of them) are dropped into Z_1.

(2) An innocent phrase "add $B(\Phi)$ to Z_1" requires some calculation to ensure that all strands from Z_1 to be blocked will indeed be blocked. This is a part of the laboratory procedure.

The method for the synthesis of blockers described under (1) above inserts *both* G and T in a given position (for a given literal), and this doubles the number of blockers for a given clause whenever such an insertion is made (which may lead to an exponential number of blockers). However, one can think of other ways for implementing "any value" occurring in a falsifier. For example, substantial progress has been made in the chemical synthesis of *universal nucleotides* that base pair with arbitrary nucleotides, see, e.g., [4]. Hence, rather than to insert in a given position both G and T, one could just insert a universal nucleotide there, and so we would have one blocker per clause. Since some of these universal nucleotides base pair more efficiently with specific nucleotides, the coding to be used for 0 and 1 could be designed accordingly.

7. An algorithm for FIND SAT

Note that SAT does not require that one finds a satisfying assignment in case that a formula Φ is satisfiable. The Find Satisfiability Problem (FIND SAT) does have this requirement. Hence, the Find Satisfiability Problem is: (1) to determine whether or not an arbitrary Boolean formula Φ is satisfiable, and (2) if Φ is satisfiable, then give an assignment satisfying Φ.

One can naturally "extend" our molecular algorithm $\mathscr{A}_1$ for solving SAT into algorithm $\mathscr{A}_2$ solving FIND SAT.

Algorithm $\mathscr{A}_2$.

Input: A Boolean formula Φ of n variables

Steps 1–5 are as in $\mathscr{A}_1$.

(6) Take a sample n-strand s

(7) Sequence s

(8) Output $asg(s)$

(9) Stop

If the algorithm $\mathscr{A}_1$ outputs "YES", then $\mathscr{A}_2$ continues in Step 6 by taking a random sample strand from the solution Z_2 which is the "end solution" of $\mathscr{A}_1$. (It should be clear now why in $\mathscr{A}_1$ we have shifted the stop instruction into a separate Step 6.)

This sample strand is sequenced (i.e., read) in Step 7, the resulting sequence is outputed in Step 8, and $\mathscr{A}_2$ stops in Step 9.

It is easy to see that algorithm $\mathscr{A}_2$ either (1) stops and outputs "NO", or (2) stops and outputs "YES φ". Case (1) holds if and only if Φ is not satisfiable, and case (2) holds if and only if Φ is satisfiable and φ is an assignment satisfying Φ.

When we take a sample n-strand in Step 6, then we really intend to pick up a double stranded DNA molecule, because one of its strands represents an assignment satisfying Φ. Because after a successful PCR such molecules are in "overwhelming majority", the probability that such a strand is picked up is very high. Moreover, one can make sure that the molecule to be eventually sequenced is a double stranded DNA molecule (e.g., by labeling all the PNA blockers, so that the double stranded PNA–DNA molecules can easily be recognized).

One could go a step further and ask for an algorithm that given a Boolean formula Φ and a positive integer k, (1) decides whether or not Φ is satisfiable, and if it is, then (2) decides whether or not there are *not less* than k assignments satisfying Φ, and if this is the case, then (3) lists k assignments satisfying Φ. Again, only blocking and PCR detection (and sequencing of sample molecules) is needed for such an algorithm.

Alternatively, using the same basic operations, one can ask for an algorithm that, given a Boolean formula Φ and a positive integer k, (1) decides whether or not Φ is satisfiable, and if it is, then (2) decides whether or not there are *exactly* k assignments satisfying Φ, and if this is the case, then (3) lists *all* k assignments satisfying Φ.

8. Discussion

Ever since Adleman's experiment [1] that has initiated by now very active research in DNA computing, the *filtering methodology* for DNA computing has been extensively studied and applied (see, e.g., [3, 6, 7, 9]). It consists of (1) the generation of the library of molecules representing all *candidate solutions* of a given computational problem,

and then (2) the *elimination*, through a sequence of biolab operations, of all these molecules from the library that do not represent *solutions*, and (3) testing whether the set of remaining molecules is non-empty. It is in particular Step 2 of this procedure that often involves a lot of lab work, and slows very much the whole process of molecular computing.

We propose in this paper the *blocking methodology*, where Step 2 above is replaced by an *inactivation* (rather than elimination) of not desired molecules, by a simple hybridization process. Thus in its pure form, the blocking process is passive—nothing is eliminated, destroyed or removed, but simply a part of the solution mixture is inactivated, so that it does not participate in the *detection* (of a solution) step of the blocking algorithm. Moreover, after the input is read (and blockers produced, or addressed, on-line), the blocking is done in one biostep.

As a matter of fact, combining the filtering and blocking methodologies may provide interesting results (at least from the theoretical point of view). Thus, e.g., one can modify the molecular algorithm for the Hamiltonian Path Problem proposed by Adleman in [1] as follows (we assume here the familiarity with [1]). After the library of paths in the input graph G has been generated by self-assembly, one proceeds with the *filtering* step that filters out all the paths of length n, where n is the number of nodes of G. But then, rather than to test whether (the coding of) each of the nodes is present exactly once (by iterating affinity separation for each node), we can add to the solution, in one step, the set of blockers which block all the molecules containing a code of the same node at least twice. Such blockers are easy to design.

We have illustrated the main principle of blocking by considering a molecular algorithm for the SAT problem. The straightforward use of PCR for the detection problem, as well as for finding solutions in the FIND SAT problem, are the advantages of the blocking principle. In a sense our solution to SAT and FIND SAT can be called either "solving by (iterating) PCR" or "the lazy man methodology": one throws to the solution mixture (containing the library, prepared a priori) the set of blockers (synthesized on-line, or fetched on-line from an accessible library of PNA molecules), presses "the PCR button", and watches for a possible increase in the number of DNA molecules, which can be made visible by using a DNA-specific dye.

It follows directly from the above discussion, that the blocking principle is different from the natural use of hybridization for the purpose of protection. Thus, e.g., in [7] the Mark operation was used to protect single stranded molecules (by hybridizing them to their complements) during the destruction process by exonucleases of those single stranded molecules that turn out not to be solutions for a given computational problem. This is a very *active* computational procedure involving a number of (iterated) biolab operations, which moreover is quite dual to the blocking principle: one manipulates (destroys) the molecules that *do not* represent solutions, and protects those that do.

Finally, we want to recall that, while the current paper is *purely theoretical* (in the sense explained in the Introduction), the companion paper [11] reporting on the experimental aspects of the blocking principle is in preparation.

Acknowledgements

The authors are indebted to H. Kusters and P. Savelkoul for the discussions on blocking already in 1998. They are also grateful to H.J. Hoogeboom, the anonymous referee, and especially to C. Henkel and K. Schmidt for useful comments on the first version of this paper. They are also indebted to Marloes van der Nat for the expert typing of the paper.

References

[1] L.M. Adleman, Molecular computation of solutions to combinatorial problems, Science 226 (1994) 1021–1024.
[2] T.A. Brown, Gene Cloning, Chapman & Hall, London, 1996.
[3] A. Condon, G. Rozenberg (Eds.), Proc. 6th Internat. Meeting on DNA Based Computers, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2001.
[4] F. Hill, D. Loakes, D.M. Brown, Polymerase recognition of synthetic oligodeoxyribonucleotides incorporating degenerate pyrimidine and purine bases, Proc. Natl. Acad. Sci. USA 95 (1998) 4258–4263.
[5] H.R. Lewis, C.H. Papadimitriou, Elements of the Theory of Computation, Prentice-Hall, Upper Saddle River, NJ, 1998.
[6] R.J. Lipton, DNA solution to hard computational problems, Science 268 (1995) 542–545.
[7] Q. Liu, L. Wang, A.G. Frutos, A.E. Condon, R.M. Corn, L.M. Smith, DNA computing on surfaces, Nature 403 (2000) 175–179.
[8] V. Manca, C. Martin-Vide, G. Paun, New computing paradigms suggested by DNA computing: computing by carving, Bio Systems 52 (1999) 47–54.
[9] G. Paun, G. Rozenberg, A. Salomaa, DNA Computing, Springer, Berlin, Heidelberg, 1998.
[10] A. Ray, B. Norden, Peptide nucleic acid (PNA): its medical and biotechnical applications and promise for the future, FASEB J. 14 (2000) 1041–1060.
[11] H. Spaink, G. Rozenberg, Experimental aspects of DNA computing by blocking, in preparation.
[12] J.D. Watson, F.H.C. Crick, Molecular structure of nucleic acids: a structure for deoxyribose nucleic acid, Nature 171 (1953) 737–738.