# A Clause String DNA Algorithm for SAT

Vincenzo Manca[1] and Claudio Zandron[2]

[1] Università degli Studi di Pisa
Dipartimento di Informatica
Corso Italia 40, 56125 Pisa, Italy
mancav@di.unipi.it
[2] Università degli Studi di Milano – Bicocca
Dipartimento di Informatica, Sistemistica e Comunicazione
Via Bicocca degli Arcimboldi 8, 20126 Milano, Italy
zandron@disco.unimib.it

**Abstract.** A DNA algorithm for SAT, the satisfiability of propositional formulae, is presented where the number of separation steps is given by the number of clauses of the instance. This represents a computational improvement for DNA algorithms based on Adleman and Lipton's *extraction model*, where the number of separations equates the number of literals of the instance.

## 1 Introduction

Since seminal Adleman's experimental DNA solution of a Directed Hamiltonian Path Problem [1], many experiments, based on molecular biology methods, were carried on to solve *hard* combinatorial problems. In fact, DNA provides a massive computational parallelism which allows us to attack combinatorial problems that, in terms of conventional computation models, are *intractable* (technically, deterministically solvable in a time that is not polynomial with respect to the *dimension* of the instances [6]). In the area of formal language theory, a great number of theoretical studies [19,16,4], related to mathematical models of DNA recombinant behavior, were inspired by this experimental possibility. Recently, in the context of DNA computing, membrane computing, and aqueous computing the combinatorial NP-*complete* problem SAT, of satisfiability for formulae of propositional logic [6], was considered by several authors, e.g. [12,9,17,3,20,21,24,8,23]. In general, in many different fields – ranging from classical combinatorial analysis to statistical physics – there is a growing interest in using SAT as a practical tool for solving real-world problems [22,7]. In this paper we apply a sort of *duality principle* which transforms the candidate solutions expressed as *Literal Strings* in [21] into *Clause Strings*. This will allow us to reduce in a remarkable way the number of more critical DNA operations necessary to solve the propositional satisfiability, according to the Adleman and Lipton's canonical model of DNA computing [5].

N. Jonoska and N.C. Seeman (Eds.): DNA7, LNCS 2340, pp. 172–181, 2002.

## 2 Propositional Satisfiability

SAT can be formulated in the following way. *Given a propositional formula $\varphi$, i.e. an instance for SAT, find if it can be satisfied for some values of its propositional variables* (*i.e.* $\varphi \in SAT$).

An equivalent formulation, directly derived by the *clause representation* of propositional formulae, can be expressed in terms of solvability of a system of boolean equations. Let us say *literal* any boolean variable or any negation of a boolean variable. Consider a system of equations over the boolean algebra of the truth values $0, 1$ with the unary operation of *negation* ($\neg$) and the binary operation of *disjunction* ($\vee$) such that $\neg 0 = 1, \neg 1 = 0, 0 \vee 0 = 0, 1 \vee 1 = 1, 0 \vee 1 = 1, 1 \vee 0 = 1$. Assume that in every equation the left member is a disjunction of literals, called a *clause*, and the right member is 1. We say *assignment* the values associated to the variables. An instance (expressed as boolean equations) belongs to SAT if there is an assignment that satisfy all equations of the system.

$$
\begin{aligned}
\neg X_3 \vee \quad X_6 \vee \quad X_8 &= 1 \\
X_2 \vee \quad X_4 \vee \quad X_8 &= 1 \\
X_3 \vee \quad X_7 \vee \quad X_{11} &= 1 \\
X_1 \vee \neg X_2 \vee \quad X_5 &= 1 \\
\neg X_5 \vee \quad X_6 \vee \quad X_{10} &= 1 \\
\neg X_3 \vee \neg X_4 \vee \neg X_{10} &= 1 \\
\neg X_4 \vee \neg X_{10} \vee \neg X_{11} &= 1 \\
X_4 \vee \neg X_5 \vee \quad X_{10} &= 1 \\
X_5 \vee \neg X_7 \vee \quad X_{11} &= 1 \\
X_3 \vee \neg X_4 \vee \neg X_9 &= 1 \\
\neg X_1 \vee \quad X_7 \vee \neg X_8 &= 1 \\
X_4 \vee \quad X_8 \vee \quad X_9 &= 1 \\
X_4 \vee \neg X_7 \vee \neg X_{10} &= 1 \\
\neg X_2 \vee \quad X_9 \vee \neg X_{11} &= 1 \\
X_1 \vee \quad X_6 \vee \neg X_8 &= 1 \\
\neg X_6 \vee \neg X_8 \vee \neg X_9 &= 1 \\
\neg X_6 \vee \neg X_8 \vee \neg X_{10} &= 1 \\
X_2 \vee \neg X_5 \vee \quad X_{11} &= 1 \\
X_1 \vee \quad X_7 \vee \quad X_{10} &= 1 \\
\neg X_1 \vee \quad X_3 \vee \neg X_9 &= 1
\end{aligned}
$$

A 3-SAT instance as a system of boolean equations

The instance considered, we call it TAMPA [14], is of type 3-SAT(11, 20) because it has 3 literals per clause, 11 variables, and 20 clauses. It derives from a 3-SAT(11, 22) instance randomly generated, that was slightly modified by deleting the clauses 5 and 12 that resulted to be tautologies (equal to 1 for any value of their variables) and by performing some other minor changes. Experiments on its DNA solvability are currently in progress at *Laboratories of Microbial Biotechnology and Environmental Microbiology* of the *Dipartimento Scientifico e Tecnologico* at the University of Verona [14].

## 3 The Extract Model

The canonical computation model of DNA computing remains so far the *extract model* of Adleman [1] as generalized by Lipton [12]. This model is based on two basic parts [5]: i) a test tube of DNA strands encoding a set of *candidate solutions* of the problem (usually obtained by annealing and ligation from an initial test tube encoding the data), and ii) a procedure that *extracts* the true solutions (the "good strands") from the non-solutions (the "bad strands"). The extraction procedure can be performed by a minimal set of DNA operations on test tubes: *separate, combine, detect*. The operation "separate" takes as input a test tube T and a sequence S, and produces as output a *yes-tube* containing the strands in T where S occurs as subsequence, and a *no-tube* containing the other strands of T. The "combine" operation takes as input two tubes and produces as output a single tube containing the strands of both the input tubes. The "detect" operation checks if the *final* test tube contains any DNA strands, and in that case it chooses one of them and determines the sequence of their nucleotides. Due to the DNA realization of these operations, the computational cost of a DNA computation can be identified with the number of separation steps.

Other operations can be incorporated in the model that are based on PCR, gel-electrophoresis, restriction enzymes, or more complex biotechnological protocols, but they can be viewed as additional tools for performing separation steps that are the *dominant operations* in a DNA computation.

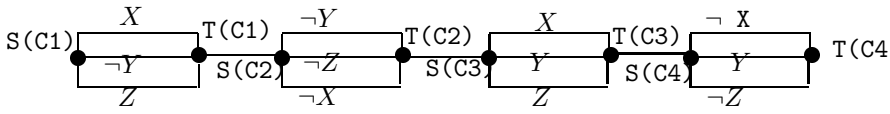
Therefore, according to the paradigm of the extraction model, two parameters are essential in the evaluation of a DNA algorithm based on this model: i) the size of the *solution space*, that corresponds to the amount of DNA necessary to encode the set of candidate solutions, and ii) the number of separation steps necessary to get the final test tube. This means that in order to improve DNA algorithms based on the extract model there are three possible ways, that might be also integrated: 1) decreasing the size of the solution space [24], 2) finding more efficient DNA implementation of the operation "separate" [3], or 3) decreasing the number of separation steps necessary to get the final test tube [20,21]. The best solution, with respect to the third point, would be some implementation of one big-step separation that could perform all the separations in a time to a great extent independent from the instance size. Papers [20] and [21] represent attempts of separation in a constant time. But in the first case a sophisticated biotechnology is necessary that needs more experimental work in order to be applied to significant examples; in the second case, as it will shown later on, the constant time for separation is obtained by increasing too much the number of candidate solutions; therefore the problem remains of finding more efficient implementations that could balance this solution space amplification.

For the further discussion it will be useful to recall the following resolution schema of the first DNA algorithm introduced by Lipton for solving SAT [12]. Encode all possible assignments by DNA *assignment strands* where, for each propositional variable $X$, a DNA oligonucleotide which encodes either $X$ or $\neg X$ occurs in the strand. When a complete pool of assignment strands is supposed

to be generated, then this pool is filtered by means of separation procedures, one for each clause. For example for the clause $(\neg X_3 \vee X_6 \vee X_8)$ the strands are separated into two tubes $A, B$. In $A$ are collected the strands where (the encoding of) $\neg X_3$ occurs, while in $B$ are collected those where $\neg X_3$ does not occur. The strands of $B$ are then separated into the tubes $C, D$. In $C$ are collected the strands where $X_6$ occurs, while in $D$ there are those where $X_6$ does not occur. Finally, the strands of $D$ are separated into $E$ and $F$, in $E$ there are the strands where $X_8$ occur and in $F$ the remaining strands of $D$. Then the strands of $A, C, E$ are merged in a test tube where the assignments which satisfy the clause are collected. In general, for any clause, the strands are kept where at least one literal of the clause occurs. This means that for a clause where $k$ literals occur we must apply $k$ separation steps. Therefore, in the case of a 3-SAT$(n, m)$ we need $3m$ separation steps.

## 4 Contact Formulations of SAT

SAT is equivalent to the *Contact Network Problem* [12]. In fact, associate to every clause $C$ a graph with two nodes, the *source* and the *target* $S(C)$, $T(C)$, and, for every literal in the clause, an edge connecting these nodes labeled by the literal. Let $X_1, X_2, \ldots, X_n$ and $C_1, C_2, \ldots, C_m$ be the variables and the clauses of our instance. Then, connect the target of $C_i$ with the source of $C_{i+1}$, for $i = 1, \ldots, m-1$.




A Contact Network Instance

The following algorithm and an ingenious DNA implementation of it, which uses DNA hairpin structures, was described in [9].

**Jonoska et al's algorithm:**
Consider many copies of a graph $G$ that is an instance for the Contact Network Problem. Separate these copies into two different almost "equivalent" pools $A, B$. Remove in all the graphs of $A$ the edges labeled by $X_1$ and in the graphs of $B$ the edges labeled by $\neg X_1$. Unite the resulting pools into a unique pool and apply the same procedure (separation, edge removing, unification) for $X_2, \ldots, X_n$. The original propositional formula can be satisfied if, at end of this process, a graph remains that connects the source of the first clause with the target of the last one.

In the following, $Lit(C)$ is the set of literals of the clause $C$, and $Cla(L)$ is the set of clauses where the literal $L$ occurs. Let us call *theory* a set of clauses and *diagram* a *coherent* set of literals (no literal and its negation can occur

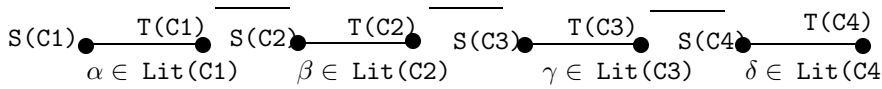
both in a diagram). Then *Lit* and *Cla* can be extended as functions from the theories to the sets of diagrams, where a natural ordering relation is given by the usual set inclusion. It is easy to verify that in this way *Lit* and *Cla* define a *Galois correspondence* (a pair of functions between ordered sets which reverse the order). This is a special case of a more general correspondence between models and theories in the first order logic, and will be the basis of our formulation of SAT given in the next section (the central role of Galois correspondences in the solvability of algebraic equations is a well known fact which was the basis of modern algebra).

The Contact Network Problem transforms easily in the following *Literal String Problem*, that is, the problem of finding paths (if any) connecting all the clauses by using a *constructive* approach rather than a destructive one, as in Jonoska et al.'s algorithm, followed by a successive *extraction* procedure.

Assume a pair of nodes, source and target $S(C), T(C)$ for any clause $C$. The following algorithm was described in [21].

**Sakamoto et al's algorithm:**
Assume a pool of elementary graphs, each one constituted by an edge labeled by a literal $L$ that connects the nodes $S(C), T(C)$ if $L$ belongs to $Lit(C)$. Then start a linking process that constructs a literal string by adding an edge between a target of a clause and the source of the next clause (w.r.t. a prefixed order of the clauses). After that, a *decimation* of the paths is performed by applying a test which controls whether a given path is *incoherent*, that is, whether a literal label occurs in both its positive and negative forms. Any literal string that results to be *incoherent* is then removed. The initial propositional formula is satisfiable if, at end, some *coherent* contact paths remain after the elimination of the incoherent ones.

The following diagram is relative to a SAT(3,4) instance.




Literal Strings

The strings: $(X\ \neg Y\ X\ Z)$, $(\neg Y\ \neg Y\ X\ Z)$, $(Z\ \neg Y\ X\ Z)$ are some coherent literal strings, in the case the clauses are those given in the contact network instance above.

It is easy to realize that for a 3-SAT instance the solution space of this algorithm is $3^m$ where $m$ is the number of clauses. A DNA solution of this algorithm has been implemented for a 3-SAT(6, 10) instance in [21]. This algorithm uses a more complex separation method that is based on hairpin formation rather than on hybridization-affinity. In this way the separation procedure can be made in a constant time (by using traditional separations, $4n$ steps would be necessary, where $n$ is the number of variables). However, although the method is really

interesting, and requires a constant time for separation, in its actual form the size of the solution space is a strong limitation for its scale up. In fact, typically the number of clauses is about 3 or 4 times the number of variables.

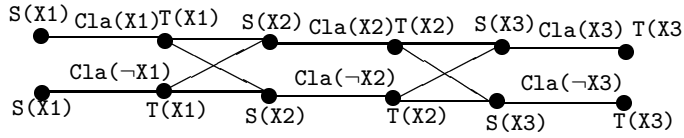
## 5 Clause String Formulation of SAT

Let us consider another *contact* formulation of the propositional satisfiability, introduced in [13], that is related to the Galois correspondence $(Lit, Cla)$ indicated in the previous section.

Consider the pair of nodes $S(X), T(X)$ (source and target) for each variable $X$. A dual perspective of the literal string formulation of SAT leads to the following algorithm.

**A clause string algorithm:**
Assume a pool of elementary graphs where $S(X), T(X)$ are connected either with an edge labeled by $Cla(X)$, or with an edge labeled by $Cla(\neg X)$ (the clauses where $X$ occurs and those where $\neg X$ occurs, respectively). Then start a linking process building the clause strings where the target of a variable is linked to the source of the next variable (according to a prefixed order of the variables).

The clause strings coincide with the paths of the graph above that connect one node of the first variable with one of the last variable. A clause string is *complete* if each clause belongs to some set that occurs as a label. Remove the clause strings that are not complete. If some complete clause strings remain, they are solutions of the considered instance of SAT.




Clause Strings

It is easy to realize that the solution space of the clause string algorithm is $2^n$ where $n$ is the number of variables. Moreover, as it will be completely clear in the following section, the number of separations that are necessary in order to extract the solutions are $m$, that is the same number of the clauses of the instance. This fact implies the possibility of a scale up in the DNA solution of SAT, independently from any DNA implementation of the separation steps.

Another advantage of representing candidate solution by means of clause strings is due to the fact that a clause is encoded several times in a clause string, as much as it can belong to $Lit(L)$ for several literals $L$ (3 in the average case, for 3-SAT). This aspect will give for free an increasing in the probability of the expected hybridizations necessary for separations, according to the analysis developed in [2] for *double encoding*.

In order to define our DNA algorithm we use a further formulation of SAT. In fact, we encode the clause string formulation of SAT in terms of the *Bipartite Covering Problem* problem (BCP) that can be stated in the following way.

Given a finite set $C$ and $n$ pairs of subsets of $C$: $A_1/B_1, \ldots, A_n/B_n$ such that $A_i \cap B_i = \emptyset$ for $i = 1, \ldots, n$, find (or at least, say if it is possible to find) $n$ sets $Y_1, \ldots, Y_n$ such that $C = Y_1 \cup \cdots \cup Y_n$, and either $Y_i = A_i$ or $Y_i = B_i$.

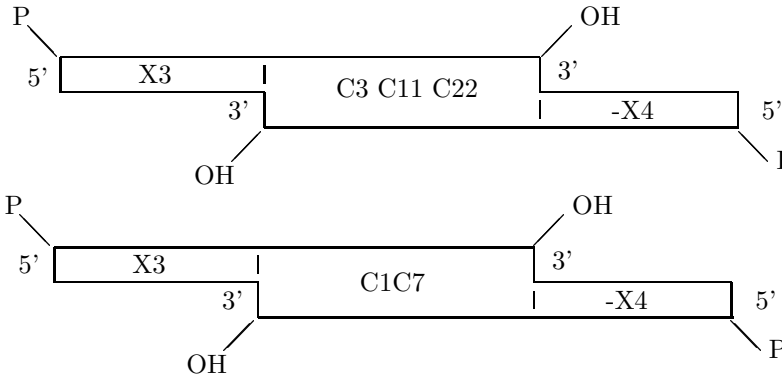
It is a simple exercise to transform any instance of SAT into an equivalent instance of BCP and viceversa. The following is the BCP formulation of the 3-SAT(11,20) instance TAMPA, already given as system of boolean equations. For the sake of brevity, we use the indexes of clauses instead of clauses, following the order of the boolean equations; however, in order to keep the original indexes of the randomly generated instance, numbers 5 and 12 are skipped (for $i = 1, \ldots, 11$, we put Ai = $Cla(Xi)$ and Bi= $Cla(\neg Xi)$).

| | | | | |
|---|---|---|---|---|
| A1/B1 | = | 4,17,21 | / | 13,22 |
| A2/B2 | = | 2,20 | / | 4,16 |
| A3/B3 | = | 3,11,22 | / | 1,7 |
| A4/B4 | = | 2,9,14,15 | / | 8,7,11 |
| A5/B5 | = | 4,10 | / | 6,9,20 |
| A6/B6 | = | 1,6,17 | / | 18,19 |
| A7/B7 | = | 3,13,21 | / | 10,15 |
| A8/B8 | = | 1,2,14 | / | 13,17,18,19 |
| A9/B9 | = | 14,16 | / | 11,18,22 |
| A10/B10 | = | 6,9,21 | / | 7,8,15,19 |
| A11/B11 | = | 3,10,20 | / | 8,16 |

The BCP Instance

## 6 The DNA Algorithm

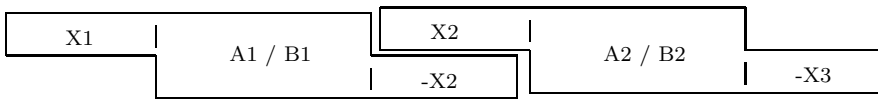
Now we encode A1, B1, ..., A11, B11 with DNA *dominoes*, that is, DNA molecules constituted, for every $i = 1, \ldots, 11$, by two different central parts corresponding to Ai and Bi, but having the same left sticky end Xi, and the same right sticky end –Xi+1. For every $i = 1, \ldots, 11$, Xi and –Xi are complementary DNA single strands. This means that we have, for any variable Xi, two dominoes with sticky ends (Xi, –Xi+1) where in the middle there are encoded either all the clauses satisfied by Xi or all the clauses satisfied by ¬Xi, respectively. For example, the dominoes for A3 and B3 have the following shape:




DNA dominoes encoding A3 and B3

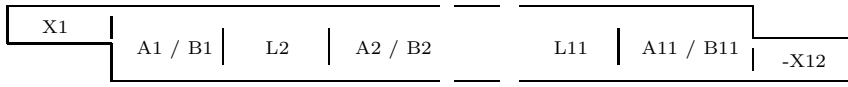
The sticky ends may correspond i) either to sites of non-palindromic restriction enzymes obtained by bigger purchased oligos after amplification and digestion with the enzymes relative to the linkers, ii) or to dominoes obtained from single stranded DNA molecules as in the original Adleman's approach (see [25] for a detailed analysis of the second method).

Now, put in a test tube many copies of the dominoes A1, B1, ..., An, Bn. After annealing and ligation of complementary sticky ends, we expect to obtain a pool of linked DNA dominoes (actually, X1 and X12 do not perform any link, but can be useful for amplification by PCR).




A DNA string with two dominoes

If we indicate by the oligo Li the pairing of the two complementary strands –Xi and Xi, which acts as a linker between two dominoes, then a long string of dominoes where all the linkers occur will have the following shape:




Linked DNA dominoes

Our algorithm is based on the validity of the following proposition that follows from the analysis so far developed.

**Proposition**
*DNA strings where all the clauses are encoded represent solutions of our instance.*

## 7 Conclusions

The proposed algorithm is really different from Lipton's method of generating assignments and then filtering them (which a lot of DNA algorithms for SAT are based on). But, it is also really different from the approach of generating $3^m$ literal strings ($m$ clauses, where typically $m$ is three or four times the number of variables) and filtering them by 'coherence' [21].

In fact, we have shown that we can generate clause strings and then filter them by 'completeness'. This leaves the dimension of the solution space the same as in Lipton's method, that is $2^n$ ($n$ variables), but in this case $m$ separations are enough, whereas the number of separations necessary in the Lipton algorithm is given by the number of literals (or the number of binary connectives plus 1). This means that in the case of a 3-SAT instance the number of dominating operations in our case is one third with respect to the algorithms based on Lipton's schema. This analysis is independent from any particular DNA implementation that our algorithm is based on. In other words, any DNA algorithm for SAT that is based on: 1) a ligation protocol of small DNA 'pieces', and 2) a separation protocol for filtering the solutions in the pool of candidate solutions, can be improved if candidate solutions express clause strings rather than assignments or literal strings. In conclusion, in the class of DNA algorithms for SAT based on Adleman and Lipton's extract model the clause strings approach implies a direct scale-up in the size of DNA solvable instances. Other more theoretical aspects of the clause string representation of SAT are related to other formulations of SAT in terms of matrices and membrane systems that are under current investigation [15] and that could turn to be relevant for DNA computing too.

## References

1. L. M. Adleman, Molecular Computation of solutions to combinatorial problems, Science, Vol. 266, pp. 1021–1024, November 11, 1994.
2. D. Boneh, C. Dunworth, R. J. Lipton, J. Sgall, Making DNA computers error resistant, in [18] pp. 163–170, 1999.
3. R. S. Brainch, C. Johnson, P.W.K. Rothemund, D. Hwang, N. Chelyapov, L. M. Adleman, Solutions of a Satisfiability Problem on a Gel-Based DNA Computer, Sixth International Meeting on DNA Based Computers, Leiden Center for Natural Computing, A. Condon G. Rozenberg (eds.), Leiden, 2000.
4. C. Calude, Gh. Păun, *Computing with Cells and Atoms*, Taylor and Francis, London, 2000.
5. K. Chen, E. Winfree, Error Correction in DNA Computing: Misclassification and Strand Loss, in [26] pp. 49–63, 2000.

6. M. R. Garey, D. S. Johnson, Computers and Intractability, Freeman, San Francisco, 1979.
7. C. P. Gomes, B. Selman, N. Crator, H. Kautz, Heavy-tailed phenomena in satisfiability and constraint satisfaction problems, J. of Automated Reasoning, Vol. 24 (1/2), pp. 67–100, 1999.
8. T. Head, X. Chen, M. J. Nichols, M. Yamamura, and S. Gal, Aqueous Solutions of Algorithmic Problems: emphasizing knights on a $3 \times 3$, in [10] pp. 219–230, 2001.
9. N. Jonoska, S. A. Karl, M. Saito, Three Dimensional DNA Structures in Computing, Biosystems, Vol. 52, 242–245, 1999.
10. N. Jonoska, N. Seeman (eds.), 7th International Meeting on DNA Based Computers, Preliminary Proceedings, Tampa, FL (U.S.A.), 2001.
11. L. Landweber, E. Baum, DNA Based Computers II, DIMACS Series 44, American Math. Society, Providence, RI, 1999.
12. R. Lipton, DNA Solutions of hard computational problems, Science, Vol. 268, 242–245, 1995.
13. V. Manca, Monoidal systems and membrane systems, in Pre-proc. Workshop on Multiset Processing, Curtea de Arges, Romania, TR 140, CDMTCS, Univ. Auckland (New Zealand), pp. 176–190, 2000.
14. V. Manca, and Di Gregorio S., Lizzari D., Vallini G., Zandron C., A DNA Algorithm for 3-SAT(11,20), in [10], pp. 167–178, 2001.
15. V. Manca, Membrane Algorithms for Propositional Satisfiability, in Pre-proc. Workshop on Membrane Computing, Curtea de Arges, Romania, TR 17, GRLMC, Univ. Rovira i Virgili, Tarragona (Spain), pp. 181–192, 2001.
16. Gh. Păun, G. Rozenberg, A. Salomaa, DNA Computing: New Computing Paradigms, Springer-Verlag, Berlin, 1998.
17. Gh. Păun, P Systems with Active Membranes: Attacking NP Complete Problems, J. Automata Languages and Combinatorics, 6, 1, 2001.
18. H. Rubin, D. Wood, DNA Based Computers III, DIMACS Series 48, American Math. Society, Providence, RI, 1999.
19. G. Rozenberg, A. Salomaa, eds., *Handbook of Formal Languages*, Springer-Verlag, Heidelberg, 1997.
20. K. Sakamoto, H. Gounzu, D. Kiga, K. Komiya, H. Gouzu, S. Yokoyama, T. Yokomori, S. Ikeda, H. Sugiyama, M. Hagiya, State transitions by molecules, Biosystems, Vol. 52, pp. 81–91, 1999.
21. K. Sakamoto, H. Gounzu, K. Komiya, D. Kiga, S. Yokoyama, T. Yokomori, M. Hagiya, Molecular Computation by DNA Hairpin Formation, Science, Vol. 288, pp. 1223–1226, May 19, 2000.
22. K. Selman, H. Kautz, B. Cohen, Local Search Strategies for Satisfiability Testing, in: Cliques, Coloring, and Satisfiability, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, D. S. Johnson, M. A. Trick, eds., vol. 26, AMS, 1996.
23. Y. Takenaka, A. Hashimoto, A proposal of DNA computing on beads and its application to SAT problems, in [10], pp. 331–339, 2001.
24. H. Yoshida and A. Suyama, Solutions to 3-SAT by breadth first search, in [26] pp. 9–22, 2000.
25. M. Yamamoto, J. Yamashita, T. Shiba, T. Hirayama, S. Takiya, K. Suzuki, M; Munekata, and A. Ohuchi, A study of hybridization process in DNA computing, in [26], pp. 101–110, 2000.
26. E. Winfree E., D. K. Gifford, DNA Based Computers V, DIMACS Series 54, American Math. Society, Providence, RI, 2000.