

Modelling DNA Computation by an Artificial Chemistry Based on Pattern Matching and Recombination

Kazuto Tominaga

Tokyo University of Technology, Hachioji, Tokyo, Japan
tomi@acm.org

Abstract

We show that a simple artificial chemistry based on pattern matching and recombination can model DNA computation in the so-called Adleman-Lipton paradigm. The paradigm exploits the huge parallelism of biochemical reactions to perform an exhaustive search. We modelled the following two methods for solving problems and obtained descriptions for them: one is the method used in Adleman's experiment, which solved a Hamiltonian path problem using actual DNA; the other is Lipton's method, designed to solve Boolean satisfiability problems. We executed the descriptions on a simulator, thereby showing the feasibility of modelling and simulating molecular computing by artificial chemistries.

Introduction

Since Adleman succeeded in solving a computational problem using actual DNA (Adleman, 1994), molecular computing using DNA has been studied energetically. The approach employed, and later extended by Lipton (Lipton, 1995), is called *the Adleman-Lipton paradigm*, which exploits the massive parallelism of biochemical reactions to perform an exhaustive search. Computation in this approach generally includes three phases: first, DNA molecules are prepared according to the problem to solve; second, they are mixed to generate solution candidates; and finally, answers are detected among them. To design and analyze DNA computation, numerous models have been proposed (Reif, 1998). Naturally most of them are built specifically for computation using DNA.

Artificial chemistries (Dittrich et al., 2001), which are usually regarded as tools for studying artificial life, could be used to model computation using DNA, since they are by definition abstract models of natural physicochemical reactions. However, only few of them have been applied for this purpose (Busch and Banzhaf, 2003).

In this paper, we try to model DNA computation using a simple artificial chemistry (Tominaga, 2004), which is not designed specifically for this purpose. The artificial chemistry uses character strings, and its dynamics are based on pattern matching and recombination. We will describe two computing methods in the Adleman-Lipton paradigm using

the artificial chemistry; the descriptions obtained will be executed by a simulator.

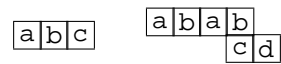
Note that we omit a great deal of biochemical details in the following discussion, so the explanations related to biochemical phenomena and experiments are not necessarily accurate.

An artificial chemistry based on pattern matching and recombination

Here we briefly explain the artificial chemistry that we are going to use in the following discussion (Tominaga, 2004).

Elements and objects

An *element* is a character; it corresponds to an atom in nature. An *object* (corresponds to a molecule) is a character string or a stack of strings, such as those depicted below.



These objects are denoted by the string notations $0:abc/$ and $0:abab/3:cd/$, respectively; the numbers represent displacements of the line relative to the first line.

Patterns

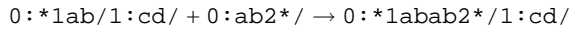
A *pattern* matches (or does not match) an object, and it can utilize two kinds of *wildcards*. An *element wildcard*, which is denoted by a number such as 1, matches any element. A *sequence wildcard*, denoted by a number and an asterisk such as 2^* and $*2$, matches any sequence of zero or more elements; the position of an asterisk represents the direction in which the sequence can extend.

A pattern is denoted in a similar way to an object. For example, the pattern $0:a1c/$ matches the object $0:abc/$; the pattern $0:*1ab/1:cd/$ matches the object $0:abab/3:cd/$. Note that the displacements are meaningful and that the length of a sequence wildcard is treated as zero in the notation for patterns.

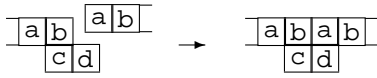
Recombination rules

A *recombination rule* transforms a group of objects into a group of objects, conserving elements just like a chemical

reaction does. It is expressed in a manner similar to chemical formulae, but in terms of patterns. An example rule is



which is illustrated as follows.



If this rule is applied to the objects $0 : abab/3 : cd/$ and $0 : abc/$, the object $0 : abababc/3 : cd/$ is produced and the reactants disappear.

Sources and drains

Objects are kept in *the working multiset*. A *source* is defined as an object, and it supplies objects of the specified form to the multiset one at a time without limit.

A *drain* is defined as a pattern, and it eliminates objects matched by the pattern, one at a time, from the multiset. A drain does not recombine objects, so wildcards in the pattern for a drain can be anonymous. Let “.” denote an anonymous element wildcard, which matches any element.

Dynamics

A *system* is a construct $\langle \Sigma, S, D, R, P_0 \rangle$ where Σ is a set of elements, S is a multiset of sources, D is a multiset of drains, R is a set of recombination rules, and P_0 is the *initial working multiset*, which specifies objects in the working multiset at the initial state. The system is interpreted nondeterministically as follows.

1. Initialize the working multiset to be P_0 .
2. Do one of the following operations.
 - Apply one recombination rule to a collection of objects.
 - Operate one source.
 - Operate one drain.
3. Go to Step 2.

Connecting multiple systems

In this paper, we newly introduce *pipes* to the artificial chemistry explained above. A pipe is a FIFO queue for objects, and it connects two systems; its entrance is a drain of one system, and its exit is a source of the other system. When the drain operates, it enqueues the object (which is to be eliminated from the system) to the pipe on its tail. The source can operate only if the pipe has at least one object. When it operates, the object at the head of the pipe is dequeued and it is supplied to the system on which the source operates.

Thus, there are two types of drains: one simply throws away objects from the system (let us call it *an open drain*), and the other feeds objects to its pipe (*connected drain*). Sources are likewise categorized into two sorts: one supplies objects to the working multiset without limit (*open source*),

and the other supplies objects that have been fed by its pipe (*connected source*).

Let us call a group of systems connected by pipes a *network*.

Basic procedure of the Adleman-Lipton paradigm

The computation in the Adleman-Lipton paradigm basically consists of the following three phases (Adleman, 1994; Lipton, 1995). All the phases are carried out using materials and methods for biochemical experiments.

- **The preparation phase** prepares single-stranded DNA molecules, each of which comprises a sequence of a small number (such as 10) of *bases*. A base is one of T: thymine, C: cytosine, A: adenine and G: guanine; T and A are *complementary* to each other, and so are C and G. The strands are designed according to the problem to be solved so that the subsequent procedure yields double-stranded DNA molecules that represent answers, if any, to the problem.
- **The assembly phase** produces double-stranded DNA molecules: two single-stranded DNA molecules that have complementary sequences of bases form a double-stranded DNA molecule. For example, a single-stranded DNA molecule that has a base sequence $5' \text{-TATAGCG-}3'$ can attach to another single-stranded DNA molecule that has a sequence $3' \text{-ATATCGC-}5'$, where $5'$ and $3'$ represent *the 5' end* and *the 3' end* of a single-stranded DNA molecule, which are chemically distinguished. Let \bar{s} denote the complementary base sequence of a sequence s .
- **The detection phase** detects DNA molecules that represent answers. DNA to be detected are specified by sequences of bases contained, or by the numbers of bases they comprise. This phase includes *amplification*, a process to increase the number of DNA molecules that have a specific base sequence, which facilitates the detection. If such a molecule is found, the problem is solved and the molecule represents an answer.

Modelling generic components of DNA computation

We are going to describe a DNA computation as a system and a network. In this section, we give descriptions that model two generic components of DNA computation.

A system for assembly

The essence of this computing paradigm is exploiting the following characteristic of DNA strands: a strand (or a part of a strand) $5' \text{-}s\text{-}3'$ attaches only to a strand (or a part of a strand) $3' \text{-}\bar{s}\text{-}5'$, where s is any sequence of bases. This reaction is called *hybridization*. The assembly phase generates candidates for an exhaustive search (in other words, a search

space), using hybridization among a huge number of single-stranded DNA molecules (approximately 3×10^{13} molecules for each edge in (Adleman, 1994)) whose sequences are designed in the preparation phase.

To make enough kinds of sequences to solve the problem, and also to reduce the possibility of erroneous reactions, a sequence is designed to comprise multiple bases, such as 10 of them. Each sequence (say $5'-s-3'$) is designed so as to hybridize only with the corresponding sequence $3'-\bar{s}-5'$, not with any part of other sequences.¹ Therefore, the sequences of bases s and \bar{s} can be regarded as *abstract bases* X and x , respectively, each of which is complementary to the other (Nishikawa et al., 2001). We assign an element to each of the sequences (i.e., abstract bases) used in the experiments discussed below. Hereafter let an uppercase letter denote an abstract base in a $5'$ to $3'$ direction, and a lowercase letter one in a $3'$ to $5'$ direction.

In the preparation phase of each of the two methods discussed subsequently, sequences are designed so that they hybridize staggered, i.e., the $3'$ end of a strand hybridizes with the $3'$ end of another strand, or a $5'$ end does with another $5'$ end. For example, DNA strands are designed to be BC and cd, so they can hybridize to form $0:BC/1:cd/$ (in our notation); this is hybridization at $3'$ ends. This molecule can hybridize further with another strand starting with D, such as DE, to form $0:BCDE/1:cd/$, which is hybridization at $5'$ ends.² Hybridization induced by the sequences at $3'$ ends of strands (say C and c) can be expressed in the artificial chemistry as the following rules.

$$\begin{aligned} 0:*1C/ + 0:c4*/ &\rightarrow 0:*1C/0:c4*/ \\ 0:*1C/0:*2/ + 0:c4*/ &\rightarrow 0:*1C/0:*2c4*/ \\ 0:*1C/ + 0:3*/-1:c4*/ &\rightarrow 0:*1C3*/0:c4*/ \\ 0:*1C/0:*2/ + 0:3*/-1:c4*/ &\rightarrow 0:*1C3*/0:*2c4*/ \end{aligned}$$

Rules for hybridization at $5'$ ends (D and d) are obtained similarly.

$$\begin{aligned} 0:*2d/ + 0:D3*/ &\rightarrow 0:D3*/0:*2d/ \\ 0:*1/0:*2d/ + 0:D3*/ &\rightarrow 0:*1D3*/0:*2d/ \\ 0:*2d/ + 0:D3*/1:4*/ &\rightarrow 0:D3*/0:*2d4*/ \\ 0:*1/0:*2d/ + 0:D3*/1:4*/ &\rightarrow 0:*1D3*/0:*2d4*/ \end{aligned}$$

We need the two sets of rules to compensate for expressive insufficiency of patterns and rules in the artificial chemistry; abstract bases have been employed for the same reason, while in (Nishikawa et al., 2001) they are introduced to enhance the performance of a simulator.

The assembly phase is modelled as a system: for each of the abstract bases to be used in the computation, the system

¹In (Adleman, 1994), the sequences of bases were created randomly, expecting that undesired hybridization, such as one between $5'-s-3'$ and $3'-\bar{t}-5'$ for $s \neq t$, would not occur.

²This process includes the concatenation of two strands BC and DE, which is called *ligation*, but we omit this biochemical detail since we are constructing an abstract model.

is equipped with rules for hybridization in the same way as shown above.

Pipes to detect specified objects

Selection with a specific sequence is performed by a pipe, i.e., a connected pair of a drain and a source. For example, a drain $0:AB1*/0:ab2*/$ models a process to detect a double-stranded DNA molecule that starts with the base sequence AB and its complement ab. The source, connected to this drain, supplies this object to another system. Selection based on lengths of DNA molecules can be performed, for example, by a drain $0:1234/0:5678/$, which extracts double-stranded objects that have exactly four bases.

Connecting systems by pipes designed in this way gives a network that models the detection phase. Amplification can also be described using the artificial chemistry, but we omit this process to focus on the algorithmic parts of the computation.

Modelling Adleman's experiment

Adleman presented a protocol to solve Hamiltonian path problems by DNA computation (Adleman, 1994). The example problem is illustrated in Figure 1; the specified start node is 0 and the end node is 6, and the only answer to this problem is the path $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6$.

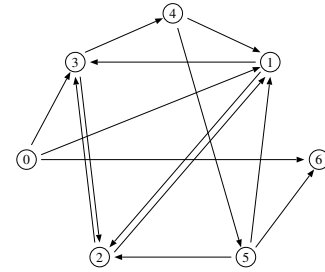


Figure 1: The problem graph (Adleman, 1994).

Preparation of DNA strands

The nodes and edges in Figure 1 are encoded as single-stranded DNA molecules as follows. We explain the encoding in terms of abstract bases.

- Each node is encoded as a strand with two bases: ab for Node 0, cd for Node 1, ..., and mn for Node 6.
- An edge from Node i to Node j is encoded as a strand with two bases, which are complementary to the sequence of the second half of Node i and the first half of Node j . For example, the edge from Node 1 (represented by cd) to Node 2 (e \bar{f}) is represented by DE. Subsequently, A is prepended to a strand that expresses an edge from Node 0 to make a fully double-stranded DNA molecule when it is obtained as an answer; N is appended to a strand that

expresses an edge to Node 6 for the same reason. (See the following picture.)

The correct answer is represented by a DNA molecule of the following form.

A	B	C	D	E	F	G	H	I	J	K	L	M	N
a	b	c	d	e	f	g	h	i	j	k	l	m	n

Assigning an element to each of the abstract bases above naturally gives objects that correspond to the designed DNA strands.

Modelling the procedure

The experimental procedure consists of the following five steps.

1. Mix single-stranded DNA designed as above together to let them hybridize.
2. Extract double-stranded DNA molecules starting with AB (i.e., Node 0) and ending with mn (Node 6).
3. Extract molecules whose lengths are exactly 14 bases.
4. Extract molecules that have all the sequences that represent Nodes 1 through 5.
5. Detect if there are any molecules.

The first step is the assembly phase and is represented as a system in the artificial chemistry, and the rest compose the detection phase and are represented as a network, as shown in Figure 2. All the systems (depicted as ovals) have the common set of elements: a to n and A to N.

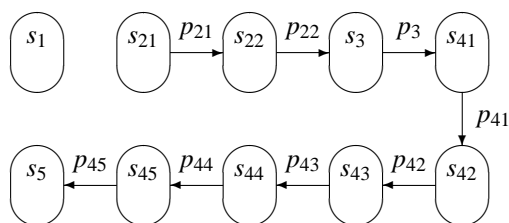


Figure 2: The system and network that model Adleman's experiment.

The system s_1 corresponds to the assembly phase (i.e., Step 1 of the procedure), and it is furnished with the set of recombination rules for hybridization. The initial multiset of s_1 comprises a number (10^5 in the example execution discussed below) of objects for each edge and node. The contents of the working multiset are moved to s_{21} (i.e., they become the initial working multiset of s_{21}) after a certain number of recombinations in s_1 .

Each of the systems consisting the network (s_{21} to s_5), which represents the detection phase, has no recombination rule, and its initial working multiset is empty except for that of s_{21} . They are connected by pipes (shown as arrows in the figure) that have the following drains.

- p_{21} : $0 : AB1^* / 0 : 2^* /$
- p_{22} : $0 : *1 / -2 : *2mn /$
- p_3 : $0 : \dots\dots\dots / 0 : \dots\dots\dots /$
- p_{41} : $0 : *12^* / 0 : *3cd4^* /$
- p_{42} : $0 : *12^* / 0 : *3ef4^* /$
- p_{43} : $0 : *12^* / 0 : *3gh4^* /$
- p_{44} : $0 : *12^* / 0 : *3ij4^* /$
- p_{45} : $0 : *12^* / 0 : *3kl4^* /$

The pipe p_{21} and p_{22} construct Step 2, p_3 corresponds to Step 3, and p_{41} through p_{45} implement Step 4. If an object appears in s_5 , it represents the answer (Step 5).

Execution

We developed a simulator for the artificial chemistry in Ruby (Matsumoto, 2001), and implemented the system and network illustrated in Figure 2. The simulator can deal with multiple systems and networks as well as moving objects from one system to another, which is the function we require between Step 1 and 2. The whole execution is performed automatically without human intervention. After approximately 18.5 hours of execution on a PC, the simulator successfully obtained nine objects in s_5 that represent the correct answer.

Modelling Lipton's method

Lipton presented a method of DNA computation to solve satisfiability problems (Lipton, 1995). The example problem is to find an assignment to the two Boolean variables x and y that satisfies the expression $(x \vee y)(x' \vee y')$, where x' and y' represent the negations of x and y , respectively. Any assignment to the two variables can be encoded as a path from the node a_1 to the node a_3 in the graph shown in Figure 3.

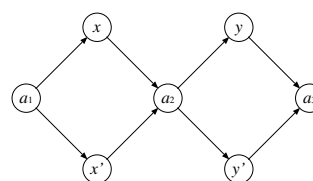


Figure 3: The graph to encode assignments (Lipton, 1995).

Preparation of DNA strands

Single-stranded DNA sequences are designed in the similar way as in Adleman's experiment: each node and edge in the graph is assigned a sequence of two abstract bases. We assign ab to the node a_1 , cd to a_2 , ef to a_3 , gh to x , ij to x' , kl to y , and mn to y' ; this assignment decides sequences for the edges (e.g., EG for the edge from a_1 to x). Objects of the form $0 : A /$ and $0 : F /$ are provided to pad the ends of double-stranded molecules.

Modelling the procedure

The experimental protocol is described in terms of test tubes (Lipton, 1995). In the following explanation, t_0, t_1 , etc. are names of the tubes.

- t_0 : tube for assembly. After DNA molecules in t_0 are hybridized to form paths in the graph (Figure 3), molecules that encode x are extracted and moved to t_1 ; the remainder are poured into t'_1 .
- t_1 : tube containing molecules that satisfy the Boolean expression x . The contents of this tube are poured into t_3 .
- t'_1 : tube containing molecules that satisfy x' . Molecules encoding y are extracted from the contents and moved to t_2 .
- t_2 : tube of molecules satisfying $x'y$. The contents are poured into t_3 and mixed with the molecules from t_1 .
- t_3 : tube of $x \vee x'y$, which is equivalent to $x \vee y$. Molecules encoding x' are extracted and moved to t_4 , and the remainder are moved to t'_4 .
- t_4 : tube of $(x \vee y)x'$. The contents are poured into t_6 .
- t'_4 : tube of $(x \vee y)x$. Molecules encoding y' are extracted and moved to t_5 .
- t_5 : tube of $(x \vee y)xy'$. The contents are poured into t_6 and mixed with the molecules from t_4 .
- t_6 : tube of $(x \vee y)(x' \vee xy')$, which is $(x \vee y)(x' \vee y')$. If there is a DNA molecule in t_6 , it represents an answer to the problem.

Each tube above is naturally mapped to a system.

Extraction of DNA based on a specific sequence is modelled as a pipe whose drain has the pattern for that sequence. Pouring all the contents to another tube is modelled as a pipe that flows any molecule to the next system. Figure 4 shows the system and network that model this method.

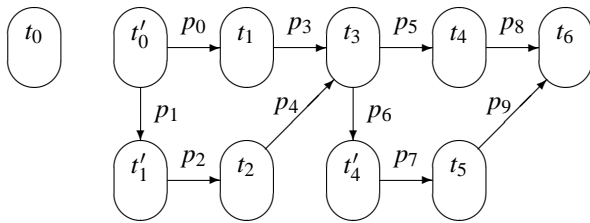


Figure 4: The system and network for solving the problem $(x \vee y)(x' \vee y')$ in Lipton's method.

The system t_0 is for the assembly phase. After a certain number of recombinations in t_0 , its contents are moved to t'_0 . (The system t'_0 does not appear as a test tube in Lipton's protocol.) The drains of the pipes that construct the network have the following patterns.

- p_0 : 0:*12*/0:*3gh4*/ (gh represents x)
- p_1 : 0:*12*/0:*3ij4*/ (ij represents x')
- p_2 : 0:*12*/0:*3kl4*/ (kl represents y)
- p_5 : 0:*12*/0:*3ij4*/ (x')
- p_6 : 0:*12*/0:*3gh4*/ (x)
- p_7 : 0:*12*/0:*3mn4*/ (mn represents y')
- p_3, p_4, p_8, p_9 : 0:*12*/0:*34*/ (flow any double-stranded molecule)

Execution

We supplied 100 objects for each node and edge in the initial working multiset of t_0 , since the search space of the example problem is tiny (2^2 possibilities) compared to Adleman's experiment. The simulator obtained objects that encode correct answers in approximately 10 seconds on a PC, including partially double-stranded objects. Depicted below are samples of objects we obtained.

A	B	G	H	C	D	M	N	E	F
a	b	g	h	c	d	m	n	e	f

encoding xy'

B	I	J	C	D	K	L	E
i	j	c	d	k	l		

encoding $x'y$

This behavior is valid: unlike Adleman's protocol, the method does not guarantee that molecules to be detected as answers are fully double-stranded.

Discussion

The artificial chemistry used to model the computing methods is general and simple, yet it describes the abstract designs of the computing processes well. For example, the system and network for Adleman's experiment illustrate that it has the assembly phase first, then the molecules are filtered through a linear procedure (Figure 2). The network for Lipton's method (Figure 4) clearly shows that the two OR operations (t'_0 to t_3 and t_3 to t_6) are concatenated by the AND operation (t_3). Thus, this reformulation of DNA computing with the artificial chemistry helps in capturing the essence of each computation, and will thereby support designing one.

Furthermore, the descriptions can be executed by the simulator. Although the current simulator can neither fully simulate biomolecular reactions that occur in the course of time, nor deal with errors (which are inevitable in biological experiments), it will show validity of a design to a certain extent, and will also help in finding unexpected behavior of the design, such as production of partially double-stranded molecules; in other words, it will help in testing and debugging a DNA computation.

In the Adleman-Lipton paradigm, the detection phase must start after sufficient candidate molecules are produced in the assembly phase. This transition cannot be modelled within the artificial chemistry, mainly because it has no notion of time. The simulator, however, implemented the transition from outside the model, i.e., iterating a recombination

a certain number of times and then moving the objects to another system. This is the reason why a computation is represented as a system and a network that are not connected.

As mentioned, we designed the two sets of rules and employed abstract bases to model hybridization to get around the expressive limitations of the artificial chemistry. We could easily extend (or specialize) patterns and recombination rules to obviate this problem by, for example, incorporating variables in patterns such as x and \bar{x} that express complementary bases. Instead of specializing the artificial chemistry by this kind of extension, we showed how a general artificial chemistry can model DNA computations. It might also be possible, with some enhancement, to model computations using more complex behavior of DNA, such as forming a hairpin structure (Sakamoto et al., 2000).

Related work

Several models have been proposed to formalize DNA computation in the Adleman-Lipton paradigm.

Reif proposed *the PAM Model* and *the RDNA Model* (Reif, 1995). PAM is an abstract model for DNA computing, and it is not necessarily suitable to model every laboratory step carried out in actual DNA computations. For example, PAM does not have an operation to express the process of selecting DNA strands of a specified length, which is naturally expressed using patterns in our artificial chemistry. RDNA, on the other hand, has the *selection* operation, which expresses this process. RDNA has the following seven basic operations: *merge*, *detect*, *separation*, *selection*, *cleavage*, *denature*, and *anneal/ligation*. These operations are specific to DNA computing. In contrast, our artificial chemistry has only three operations, which are independent from DNA computing: pattern matching/recombination, operating sources, and operating drains. Thus, our artificial chemistry is more simple and general than the RDNA Model.

Adleman proposed three abstract models, namely, *the unrestricted model*, *the restricted model*, and *the memory model* (Adleman, 1996). *DNA-EC* (Yokomori and Kobayashi, 1999) is also an abstract model for DNA computing; it describes a DNA computation in terms of set operations such as union. Since all of these are abstract models, each laboratory step does not necessarily correspond to an operation in the models. For example, Adleman's models cannot describe selection of DNA strands based on its length; DNA-EC cannot naturally express this selection using its operations. Our artificial chemistry, by contrast, naturally gave a description for each laboratory step.

The stickers model (Roweis et al., 1999) is designed for a specific computational method that uses short DNA strands called *stickers*. *YAC* (Yokomori, 2000) is also an abstract model for a specific computational method that can utilize DNA with complex structures. Although our current approach can deal only with single- and double-stranded DNA molecules, our artificial chemistry is more general than these

models in the sense that its application is not limited to a particular computing method.

Most of the models discussed above have been built particularly for DNA computation, while extension for modelling proteins or other materials is suggested (Reif, 1995). On the other hand, our artificial chemistry can model DNA computation as well, even though it was not designed specifically for this purpose. This generality of the artificial chemistry will possibly benefit modelling interconnection between DNA-computing systems and other systems that are constructed using nanotechnology. As for computational power, our artificial chemistry is equivalent to Turing machines (Tominaga, 2004), as some of other models are.

Since our simulator was not designed to simulate real biomolecular reactions, and the whole systems (including the recombination rules) were by no means refined to obtain a good performance, the result of the simulation does not have quantitative accuracy compared to the simulators specifically designed for DNA computation such as (Nishikawa et al., 2001). However, it was shown that a level of simulation using a general artificial chemistry is feasible.

Among artificial chemistries, RESAC was applied to analyze some of the quantitative properties of systems that model Adleman's experiment (Busch and Banzhaf, 2003). The problem graph (Figure 1) is encoded as clauses of the first-order predicate logic, and the clauses react with each other to solve the problem. In this modelling, correspondence between molecules in the artificial chemistry (i.e., clauses) and DNA molecules is not clear, while it is straightforward in the modelling of the present paper.

Conclusions

Artificial chemistries are promising candidates for tools to design, test and debug molecular computation. We demonstrated this by modelling DNA computation using a simple artificial chemistry, and by executing the obtained descriptions by a simulator. The descriptions are simple and straightforward, and they correspond well with laboratory steps carried out in actual DNA computations. We envisage artificial chemistries as useful vehicles not only in studying artificial life but also in designing and comprehending various molecular systems.

References

- Adleman, L. M. (1994). Molecular computation of solutions to combinatorial problems. *Science*, 266(5187):1021–1024.
- Adleman, L. M. (1996). On constructing a molecular computer. In *DNA Based Computers*, volume 27 of *DI-MACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 1–21.
- Busch, J. and Banzhaf, W. (2003). How to program artificial chemistries. In *Advances in Artificial Life (Proceedings*

of the 7th European Conference, ECAL 2003), number 2801 in LNAI, pages 20–30. Springer.

Dittrich, P., Ziegler, J., and Banzhaf, W. (2001). Artificial chemistries — a review. *Artificial Life*, 7:225–275.

Lipton, R. J. (1995). DNA solution of hard computational problems. *Science*, 268(5210):542–545.

Matsumoto, Y. (2001). *Ruby In A Nutshell*. O'Reilly & Associates.

Nishikawa, A., Yamamura, M., and Hagiya, M. (2001). DNA computation simulator based on abstract bases. *Soft Computing*, 5(1):25–38.

Reif, J. H. (1995). Parallel molecular computation. In *Proceedings of the 7th Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA '95*, pages 213–223.

Reif, J. H. (1998). Paradigms for biomolecular computation. In *Unconventional Models of Computation*, Springer Series in Discrete Mathematics and Theoretical Computer Science, pages 72–93. Springer.

Roweis, S., Winfree, E., Burgoyne, R., Chelyapov, N. V., Goodman, M. F., Rothmund, P. W. K., and Adleman, L. M. (1999). A sticker based model for DNA computation. In *DNA Based Computers II*, volume 44 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 1–27.

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

Tominaga, K. (2004). A formal model based on affinity among elements for describing behavior of complex systems. Technical Report UIUCDCS-R-2004-2413, Department of Computer Science, University of Illinois at Urbana-Champaign.

Yokomori, T. (2000). YAC: Yet another computation model of self-assembly. In *DNA Based Computers V*, volume 54 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 153–167.

Yokomori, T. and Kobayashi, S. (1999). DNA-EC: A model of DNA-computing based on equality checking. In *DNA Based Computers III*, volume 48 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 347–360.