Claudio Ferretti
Giancarlo Mauri
Claudio Zandron (Eds.)

LNCS 3384

DNA Computing

10th International Workshop
on DNA Computing, DNA10
Milan, Italy, June 2004, Revised Selected Papers

Springer

Claudio Ferretti Giancarlo Mauri
Claudio Zandron (Eds.)

DNA Computing

10th International Workshop
on DNA Computing, DNA10
Milan, Italy, June 7-10, 2004
Revised Selected Papers

Springer

Volume Editors

Claudio Ferretti
Giancarlo Mauri
Claudio Zandron
Università degli Studi di Milano-Bicocca
Dipartimento di Informatica, Sistemistica e Comunicazione
via Bicocca degli Arcimboldi 8, 20126, Milano, Italy
E-mail: {ferretti,mauri,zandron}@disco.unimib.it

Library of Congress Control Number: 2005927141

CR Subject Classification (1998): F.1, F.2.2, I.2.9, J.3

ISSN	0302-9743
ISBN-10	3-540-26174-5 Springer Berlin Heidelberg New York
ISBN-13	978-3-540-26174-2 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11493785 06/3142 5 4 3 2 1 0

Preface

Biomolecular computing has emerged as an interdisciplinary field that draws together chemistry, computer science, mathematics, molecular biology, and physics. Our knowledge of DNA nanotechnology and biomolecular computing increases dramatically with every passing year. The International Meeting on DNA Computing has been a forum where scientists with different backgrounds, yet sharing a common interest in biomolecular computing, meet and present their latest results. Continuing this tradition, the 10th International Meeting on DNA Computing (DNA10) focused on the current experimental and theoretical results with the greatest impact.

The meeting took place at the University of Milano-Bicocca, Milan, Italy, from June 7 to June 10, 2004, and it was organized by the University of Milano-Bicocca and the Department of Informatics of the University of Milano-Bicocca. Papers and poster presentations were sought in all areas that relate to biomolecular computing, including (but not restricted to): demonstrations of biomolecular computing (using DNA and/or other molecules), theoretical models of biomolecular computing, biomolecular algorithms, computational processes in vitro and in vivo, analysis and theoretical models of laboratory techniques, biotechnological and other applications of DNA computing, DNA nanostructures, DNA devices such as DNA motors, DNA error evaluation and correction, in vitro evolution, molecular design, self-assembled systems, nucleic acid chemistry, and simulation tools.

Authors were asked to choose between two different tracks:

Track A — Full paper, for authors who wished to submit a full paper for presentation at DNA10 (oral or poster), and publication in the conference proceedings.

Track B — One-page abstract, for authors submitting experimental results, and who planned to submit their manuscript to a scientific journal, rather than publish it in the conference proceedings.

We received 67 submissions in track A and 27 in track B. Among them, 30 papers were selected for oral presentation. About 140 people attended the meeting.

The first day of the meeting, June 7, 2004, was dedicated to the following tutorials: N. Pavelka (Univ. of Milano-Bicocca), "Gene Expression Studies Using Microarrays," H.J. Hoogeboom (Leiden University), "Basic Concepts of Computing for Biologists," C. Henkel (Leiden University), "Basic Molecular Biology for Nonspecialists," and T.H. LaBean (Duke University), "Self-Assembly."

The next three days were devoted to invited plenary lectures and regular oral presentations. The invited plenary lectures were by K. Benenson (Weizmann Institute of Science, Israel), "An Autonomous Molecular Computer for Logical Control of Gene Expression," C. Flamm (University of Vienna, Austria), "Computational Design of Multi-stable Nucleic Acid Sequences," G. Păun (Institute of Mathematics of the Romanian Academy, Romania), "Membrane Computing — Power and Efficiency. An Overview," J. Reif (Duke University, USA), "DNA-Based Nano-engineering: DNA and Its Enzymes as the Engines of Creation at the Molecular Scale," and W.M. Shih (Harvard University, USA), "Clonable DNA Nanotechnology."

The editors would like to thank all contributors to and participants in the DNA10 conference, the Program Committee (A. Carbone, J. Chen, N. Jonoska, L. Kari, C. Mao, G. Mauri, G. Păun, J. Rose, P. Rothemund, Y. Sakakibara, N. Seeman, E. Shapiro, L. Smith, R. Weiss, and H. Yan), and the external reviewers.

Finally, we wish to thank Brainspark plc. Comerson, the Department of Informatics, Systems and Communications of the University of Milano-Bicocca, Etnoteam, the European Commission, STMicroelectronics, and the University of Milano-Bicocca for the support and sponsorship of the conference.

January 2005

Claudio Ferretti,
Giancarlo Mauri,
Claudio Zandron

Table of Contents

Computing by Observing Bio-systems: The Case of Sticker Systems
Artiom Alhazov, Matteo Cavaliere 1

DNA-Based Computation Times
Yuliy Baryshnikov, Ed Coffman, Petar Momčilović 14

Computing Beyond the Turing Limit Using the H Systems
Cezar Câmpeanu, Andrei Păun 24

Biomolecular Implementation of Computing Devices with Unbounded Memory
Matteo Cavaliere, Nataša Jonoska, Sivan Yogev, Ron Piran, Ehud Keinan, Nadrian C. Seeman 35

Characterization of Non-crosshybridizing DNA Oligonucleotides Manufactured *In Vitro*
Junghuei Chen, Russell Deaton, Max H. Garzon, Jin Woo Kim, David Wood, Hong Bi, Dylan Carpenter, Yu-Zhen Wang 50

Error-Free Self-assembly Using Error Prone Tiles
Ho-Lin Chen, Ashish Goel 62

On the Computational Complexity of *P* Automata
Erzsébet Csuhaj-Varjú, Oscar H. Ibarra, György Vaszil 76

A Weighted Insertion-Deletion Stacked Pair Thermodynamic Metric for DNA Codes
Arkadii G. D'yachkov, Anthony J. Macula, Wendy K. Pogozelski, Thomas E. Renz, Vyacheslav V. Rykov, David C. Torney 90

DNA Extraction by XPCR
Giuditta Franco, Cinzia Giagulli, Carlo Laudanna, Vincenzo Manca 104

A Method of Error Suppression for Self-assembling DNA Tiles
Kenichi Fujibayashi, Satoshi Murata 113

Using Automated Reasoning Systems on Molecular Computing
Carmen Graciani Díaz, Mario J. Pérez-Jiménez 128

Parallelism in Gene Assembly
Tero Harju, Chang Li, Ion Petre, Grzegorz Rozenberg 138

Splicing Systems for Universal Turing Machines
Tero Harju, Maurice Margenstern 149

Application of Mismatch Detection Methods in DNA Computing
Christiaan V. Henkel, Grzegorz Rozenberg, Herman P. Spaink 159

Bond-Free Languages: Formalizations, Maximality and Construction Methods
Lila Kari, Stavros Konstantinidis, Petr Sosík 169

Preventing Undesirable Bonds Between DNA Codewords
Lila Kari, Stavros Konstantinidis, Petr Sosík 182

Testing Structure Freeness of Regular Sets of Biomolecular Sequences
Satoshi Kobayashi 192

Minimum Basin Algorithm: An Effective Analysis Technique for DNA Energy Landscapes
Mitsuhiro Kubota, Masami Hagiya 202

Efficient Initial Pool Generation for Weighted Graph Problems Using Parallel Overlap Assembly
Ji Youn Lee, Hee-Woong Lim, Suk-In Yoo, Byoung-Tak Zhang, Tai Hyun Park 215

Partial Words for DNA Coding
Peter Leupold 224

Accepting Hybrid Networks of Evolutionary Processors
Maurice Margenstern, Victor Mitrana, Mario J. Pérez-Jiménez 235

Building the Components for a Biomolecular Computer
Clint Morgan, Darko Stefanovic, Cristopher Moore, Milan N. Stojanovic 247

Methods for Manipulating DNA Molecules in a Micrometer Scale Using Optical Techniques
Yusuke Ogura, Takashi Kawakami, Fumika Sumiyama, Satoru Irie, Akira Suyama, Jun Tanida 258

From Cells to Computers: Membrane Computing – A Quick Overview
Gheorghe Păun 268

The Capacity of DNA for Information Encoding
Vinhthuy Phan, Max H. Garzon 281

Compact Error-Resilient Computational DNA Tiling Assemblies
John H. Reif, Sudheer Sahu, Peng Yin 293

Toward “Wet” Implementation of Genetic Algorithm for Protein Engineering
Kensaku Sakamoto, Masayuki Yamamura, Hiroshi Someya 308

Programmable Control of Nucleation for Algorithmic Self-assembly
Rebecca Schulman, Erik Winfree 319

DNA Hybridization Catalysts and Catalyst Circuits
Georg Seelig, Bernard Yurke, Erik Winfree 329

Complexity of Self-assembled Shapes
David Soloveichik, Erik Winfree 344

Aqueous Computing with DNA Hairpin-Based RAM
Naoto Takahashi, Atsushi Kameda, Masahito Yamamoto, Azuma Ohuchi 355

A Programmable Molecular Computer in Microreactors
Danny van Noort 365

Combinatorial Aspects of Minimal DNA Expressions
Rudy van Vliet, Hendrik Jan Hoogeboom, Grzegorz Rozenberg 375

A Design for Cellular Evolutionary Computation by Using Bacteria
Kenichi Wakabayashi, Masayuki Yamamura 389

An Inexpensive LED-Based Fluorometer Used to Study a Hairpin-Based DNA Nanomachine
Hanwen Yan 399

Designs of Autonomous Unidirectional Walking DNA Devices
Peng Yin, Andrew J. Turberfield, John H. Reif 410

Design of an Autonomous DNA Nanomechanical Device Capable of Universal Computation and Universal Translational Motion
Peng Yin, Andrew J. Turberfield, Sudheer Sahu, John H. Reif 426

A Clocked DNA-Based Replicator
Bernard Yurke, David Zhang 445

A Bayesian Algorithm for In Vitro Molecular Evolution of Pattern Classifiers
Byoung-Tak Zhang, Ha-Young Jang 458

Author Index 469

Computing by Observing Bio-systems: The Case of Sticker Systems

Artiom Alhazov[1,2] and Matteo Cavaliere[3]

[1] Research Group on Mathematical Linguistics,
Rovira i Virgili University,
Pl. Imperial Tàrraco 1, 43005 Tarragona, Spain
`artiome.alhazov@estudiants.urv.es`

[2] Institute of Mathematics and Computer Science,
Academy of Sciences of Moldova,
Str. Academiei 5, Chişinău, MD 2028, Moldova
`artiom@math.md`

[3] Department of Computer Science and Artificial Intelligence,
University of Sevilla,
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
`martew@inwind.it`

Abstract. A very common approach in chemistry and biology is to observe the progress of an experiment, and take the result of this observation as the final output. Inspired by this, a new approach to computing, called system/observer, was introduced in [3].

In this paper we apply this strategy to sticker systems, [8, 11]. In particular we use finite automata (playing the role of observer) watching the "evolution" of a sticker system and translating such "evolution" into a readable output.

We show that this way of "computing by observing" brings us results quite different from the ones obtained when considering sticker systems in the standard manner. Even regular simple sticker systems (whose generative power is subregular) become universal when considered in this new framework. The significance of these results for DNA computing (by sticker systems) is briefly discussed.

1 Introduction: Observing Sticker Systems

A usual procedure in chemistry and biology is to observe the progress of an experiment, taking the result of observation as the output. Inspired by this a new approach to computing, called system/observer, has been introduced in [3].

There it was shown how a computing device can be constructed using two less powerful systems: the first one, which is a mathematical model of a biological system, "lives" (evolves), passing from one configuration to the next, producing in this way a "behavior"; the second system, called "observer", is placed outside and watches the biological system. Following a set of specific rules the observer translates the behavior of the underlying system into a "readable" output: it associates a label to each configuration of the bio-system and writes these labels according to their chronological order onto an output tape; in this way the pair composed by the biological system and the observer can be considered a computing (generating) device, as described in Figure 1.

This idea recalls a discussion by G. Rozenberg and A. Salomaa in [12]. They remarked that the result of a computation can be seen as already present in nature: we only need to look (in an appropriate way) at it. In their case this observation is made applying a (generalized) finite state sequential transducer to the so-called twin-shuffle language, a language closely related to the structure of DNA molecules. In our case the observer is applied not only to the final result, but to the entire evolution of the system. In other words, in our architecture, the computation is made by observing the full “life” of a biological system.

Until now, the system/observer architecture has been applied in different frameworks; in the first work, [3], the evolution of a membrane system (a formal model inspired by the functioning of the living cells) has been observed. In that paper it has been shown how the system composed of a “not powerful” membrane-system (with context-free power) and a finite state automaton in the role of observer, is universal. This can be considered the first (surprising) “hint” of the fact that computing by observing is a very powerful approach.

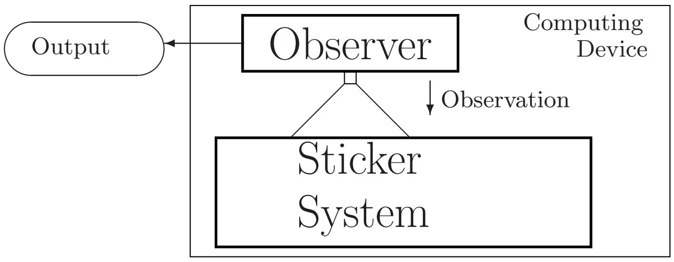


Fig. 1. Conceptual view of a sticker-system/observer architecture

In [5], a finite automaton observes the evolution of “marked” strings of a splicing system (a formal system inspired by the recombination of DNA strands that happens under the action of restriction enzymes). Also in this case, the observation adds much power to the considered bio-system. In particular, it has been shown that just observing the evolution of marked strings in a splicing system (using finite axioms and rules) it is even possible to obtain non-recursive languages (we recall that the generative power of this class of splicing systems, considered in the standard way, is subregular).

Finally, a more general application of the system/observer framework has been presented in [4]: the “evolution” of a grammar has been observed using a finite automaton. In this case, the universality is obtained using a finite state automaton observing a context-free grammar.

Here, we investigate *observable sticker-systems*, where the bio-system is a sticker system.

The main reason for investigating sticker systems in the system/observer framework comes from the fact that, using a recent lab-technique named *FRET*, [9], it is possible, under biologically relevant conditions, to observe the dynamics of a single molecule. Therefore we believe that it is extremely interesting to investigate how much we can compute just by observing the evolution of DNA molecules and sticker systems might represent an optimal way to formalize this investigation.

Sticker systems were introduced in [8] as a formal model of the operation of *annealing* (and *ligation*) operation that is largely used in DNA computing area, since the successful experiment of L.M. Adleman in 1994, [1]. The basic operation of a sticker system is the *sticking* operation that constructs double stranded sequences out of "DNA dominoes" (*polyominoes*) that are sequences with one or two sticky ends, or single stranded sequences, attaching to each other by *ligation* and *annealing*.

The informal idea of an observable sticker system can be expressed in the following way: an observer (for example, a microscope) is placed outside the "test tube", where (an unbounded number of copies of) DNA strands and DNA dominoes are placed together. Some of these molecules are marked (for example, with a fluorescent particle). The molecules in the solution will start to self-assemble (to stick to each other) and, in this way, new molecules are obtained. The observer watches the evolution of the marked molecules and stores such evolution on an external tape in a chronological order.

For each possible "evolution" of the marked molecules a certain string is obtained. Collecting all the possible "evolutions" of such marked strands we obtain a language.

Many different variants of sticker systems can be considered, using different kinds of dominoes and different restrictions on the sticking operation (see details in [11]). In this paper we consider a very restricted and simple variant of sticker system, whose power is subregular, and we show that, when we consider such variant in the system/observer framework, then we get much more generative power and even universality.

2 Formal Language Pre-requisites

In what follows we suppose the reader familiar with basic notions of formal languages (as introduced, for instance, in [13]).

We will denote a finite set (the alphabet) by V, the set of words over V by V^*. For $x \in V^*$, $Pref(x) = \{y \in V^* \mid x = yz_2, z_2 \in V^*\}$, $Suff(x) = \{y \in V^* \mid x = z_1y, z_1 \in V^*\}$ and $Sub(x) = \{y \in V^* \mid x = z_1yz_2, z_1, z_2 \in V^*\}$ are the sets of all prefixes, suffixes and subwords of x, respectively.

A *shuffle* of words $x_1 \in T_1^*$ and $x_2 \in T_2^*$ ($T_1 \cap T_2 = \emptyset$) is a word $y \in (T_1 \cup T_2)^*$ such that $h_{T_i}(y) = x_i$, $i \in \{1, 2\}$, where h_{T_i} are the projection morphisms: $h_{T_i}(a) = a$ if $a \in T_i$ and λ otherwise, for $i \in \{1, 2\}$.

By CF, CS, and RE we denote the classes of languages generated by context-free, context-sensitive, and unrestricted grammars respectively.

We shortly recall the basic notions of a *conditional grammar* used in the following theorem (for more details the reader can consult [6]).

A (context-free) *conditional grammar* is a construct $G = (N, T, P, S)$, where N and T are nonterminal and terminal symbols, S is the axiom and P is a finite set of rules of the form $(A \to \alpha, R)$, where $A \in N$, $\alpha \in (N \cup T)^*$ and R is a regular language over $N \cup T$. We say that $uAv \Rightarrow u\alpha v$ if there is a rule $(A \to \alpha, R) \in P$ such that $uAv \in R$.

For every language $L \in RE$ there exists a conditional grammar generating L. Without restricting generality we can assume that the rules of the grammar are binary ($|\alpha| \leq 2$ for every $(A \to \alpha, R) \in P$).

3 Preliminaries: Sticker Systems

We recall the basic notions of sticker systems. As it was already mentioned in the introduction, sticker systems can be considered a formal (language) model inspired by the annealing and ligation operations. The basic idea is to have initially DNA strands, called axioms, and dominoes that are DNA strands with sticky ends. Starting from the axioms and iteratively using the operation of sticking, complete double stranded sequences are obtained.

The collection of all the complete double stranded sequences obtained is the language generated by the sticker system.

Consider a symmetric relation $\rho \subseteq V \times V$ over V (of *complementarity*). Following [11], we associate with V the monoid $V^* \times V^*$ of pairs of strings. Because it is intended to represent DNA molecules, we also write elements $(x_1, x_2) \in V^* \times V^*$ in the form $\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ and $V^* \times V^*$ as $\begin{pmatrix} V^* \\ V^* \end{pmatrix}$. We denote by $\begin{bmatrix} V \\ V \end{bmatrix}_\rho = \{\begin{bmatrix} a \\ b \end{bmatrix} \mid a, b \in V, (a, b) \in \rho\}$ the set of *complete double symbols*, and $WK_\rho(V) = \begin{bmatrix} V \\ V \end{bmatrix}_\rho^*$ is the set of the *complete double-stranded sequences (complete molecules)* also written as $\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$, where x_1 is the *upper strand* and x_2 is the *lower strand.*

As in [11], we use *single strands* – the elements of $S(V) = \begin{pmatrix} \lambda \\ V^* \end{pmatrix} \cup \begin{pmatrix} V^* \\ \lambda \end{pmatrix}$ and the molecules with (a possible) *overhang* on the right, which are the elements of $R_\rho(V) = \begin{bmatrix} V \\ V \end{bmatrix}_\rho^* S(V)$, from now on called *well-started* molecules (upper and lower strand are defined as in the case of complete molecules).

Given a well started molecule $u \in R_\rho(V)$ and a single strand $v \in S(V)$, we recall in Figure 2 the partial operation $\mu : R_\rho(V) \times S(V) \longrightarrow R_\rho(V)$ of sticking, as defined in [11]. We point out that we use a case of sticking, restricted to pasting a single strand to the right side of a well-started molecule (with a possible overhang on the right), corresponding to the *simple regular* sticker systems. Furthermore, we define length of a single strand

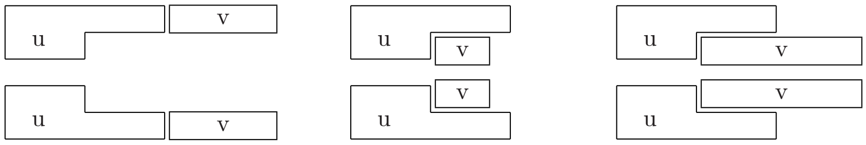


Fig. 2. Sticking operation

$u = \binom{x}{\lambda}$ (or $u' = \binom{\lambda}{x}$) as $|u| = |u'| = |x|$, and for a finite $H \subseteq S(V)$ we say $length(H) = max\{|u| \mid u \in H\}$.

A (simple regular) sticker system is a construct $\gamma = (V, \rho, A, D)$, where $A \subseteq R_\rho(V)$ is the (finite) set of axioms, and $D \subseteq S(V)$ is the (finite) set of *dominoes* (in this case these are single strands). Given $u, w \in R_\rho(V)$, we write $u \Rightarrow w$ iff $w = \mu(u, v)$ for some $v \in D$. A sequence $(w_i)_{1 \le i \le k} \subseteq R_\rho(V)$ is called a *complete computation* if $w_1 \in A$, $w_i \Rightarrow w_{i+1}$ for $1 \le i < k$ and $w_k \in WK_\rho(V)$.

The *language* generated by a sticker system γ is the set of upper strands of all complete molecules derived from the axioms. We remark the fact that the *family of languages generated by simple regular sticker systems is strictly included in the family of regular languages* (see [11] for the proof).

4 The Observer: Automata with Singular Output

For the observer (the "microscope") as described in the introduction we need a device mapping DNA molecules (also incomplete) into just one symbol.

For an alphabet V, our *double-symbol alphabet* constructed over V is

$$V_d = \begin{bmatrix} V \\ V \end{bmatrix}_\rho \cup \binom{V}{\lambda} \cup \binom{\lambda}{V}.$$

Therefore, following the idea also used in [3], we define a variant of finite state automata: the states are labeled by the symbols of the output alphabet Σ or with λ. Any computation of the automaton produces as output the label of the state it halts in (we are not interested in accepting computations and therefore do not consider the final states); because the observation of a certain string should always lead to a fixed result, we consider here only deterministic and complete automata.

An automaton with a singular output reads a molecule (element of $R_\rho(V)$) and outputs one symbol. Every well-started molecule in $R_\rho(V) \subseteq V_d^*$ is read, in a classical way, from left to right, scanning one double symbol from V_d at a time.

Formally, an automaton with singular output is a tuple $O = (Z, V_d, \Sigma, z_0, \delta, \sigma)$ with a state set Z, input alphabet V_d, initial state $z_0 \in Z$, and a complete transition function δ as known from conventional finite automata , that maps elements of $(V_d \times Z)$ into Z. Furthermore, there is the output alphabet Σ and a labeling function $\sigma : Z \longrightarrow \Sigma \cup \{\lambda\}$.

For a molecule $w \in R_\rho(V)$ and an automaton O we write $O(w)$ to indicate such output; for a sequence $w_1, \dots, w_n$ of $n \geq 1$ of molecules in $R_\rho(V)$ we write $O(w_1, \dots, w_n)$ for the string $O(w_1) \cdots O(w_n)$. For simplicity, in what follows, we present only the mapping defined by the observer without giving its real implementation as a finite automaton.

Moreover, we will also want the observer to be able to reject some words. To do this we simply choose a special symbol $\perp \notin \Sigma$ and an extended output alphabet $\Sigma_\perp = \Sigma \cup \{\perp\}$; σ then is a mapping from the set of states Z to $\Sigma_\perp \cup \{\lambda\}$. If a "bad" (not of interest) molecule is observed, then $\perp$ is produced and thus the entire sequence is to be rejected (hence, the criterion of rejecting is exactly the regular language, recognized by a finite automaton like O described above, but without output and with a single final state $\perp$). Then, using the intersection with the set Σ^*, it is possible to filter out the strings which contain $\perp$.

5 Observable Sticker Systems

An *observable sticker system* with output alphabet Σ is a construct $\phi = (\gamma, O)$, where γ is the sticker system with alphabet V, and O is the observer with input alphabet V_d constructed over V and with output alphabet Σ.

We denote the collection of all complete computations of ϕ by $\mathcal{C}(\phi)$. The language, over the output alphabet Σ, generated by an observable sticker system ϕ, is defined as $L(\phi) = \{O(s) \mid s \in \mathcal{C}(\phi)\}$. If we want to filter out the words that contain the special symbol $\perp$, then we consider the language $\widehat{L}(\phi) = L(\phi) \cap \Sigma^*$.

Here is a simple example that illustrates how an observable sticker system works. At the same time this example shows how one can construct an observable sticker system generating a non regular language (despite the fact that the power of simple regular sticker systems, when considered in the classical way, is subregular). Consider the following observable sticker system $\phi = (\gamma, O)$:

$$\gamma = (V = \{a, \mathbf{c}, \mathbf{g}, t\}, \rho = \{(a,t), (\mathbf{c},\mathbf{g}), (t,a), (\mathbf{g},\mathbf{c})\}, A = \{\begin{bmatrix} a \\ t \end{bmatrix}\}, D),$$

$$D = \{\begin{pmatrix} a \\ \lambda \end{pmatrix}, \begin{pmatrix} \lambda \\ t \end{pmatrix}, \begin{pmatrix} \mathbf{c} \\ \lambda \end{pmatrix}, \begin{pmatrix} \lambda \\ \mathbf{g} \end{pmatrix}\},$$

with the observer O defined by the following mapping:

$$O(w) = \begin{cases} b, & \text{if } w \in \begin{bmatrix} a \\ t \end{bmatrix}^* \begin{pmatrix} a^* \\ \lambda \end{pmatrix} \cup \begin{pmatrix} \lambda \\ t^* \end{pmatrix}, \\ d, & \text{if } w \in \begin{bmatrix} a \\ t \end{bmatrix}^* \begin{pmatrix} a^*\mathbf{c} \\ \lambda \end{pmatrix} \cup \begin{pmatrix} \lambda \\ t^*\mathbf{g} \end{pmatrix}, \\ \lambda, & \text{otherwise.} \end{cases}$$

The language generated by γ is $L_1 = \{b^m d^n \mid m \geq n, m \geq 1, n \geq 0\} \notin REG$.

Below is an example of computation of ϕ (generating *bbbbdd*):

Step	0	1	2	3	4	5	6
Added		$\binom{a}{\lambda}$	$\binom{a}{\lambda}$	$\binom{\lambda}{t}$	$\binom{\mathbf{c}}{\lambda}$	$\binom{\lambda}{t}$	$\binom{\lambda}{\mathbf{g}}$
Molecule	$\begin{matrix}a\\t\end{matrix}$	$\begin{matrix}aa\\t\end{matrix}$	$\begin{matrix}aaa\\t\end{matrix}$	$\begin{matrix}aaa\\tt\end{matrix}$	$\begin{matrix}aaa\mathbf{c}\\tt\end{matrix}$	$\begin{matrix}aaa\mathbf{c}\\ttt\end{matrix}$	$\begin{matrix}aaa\mathbf{c}\\ttt\mathbf{g}\end{matrix}$
Output	b	b	b	b	d	d	λ

The idea of the system ϕ is the following: think of symbols $\mathbf{c}$, $\mathbf{g}$ as "markers". While we stick to the current molecule either $\binom{a}{\lambda}$ or $\binom{\lambda}{t}$, the observer maps the result (a molecule without markers) to b. As soon as we attach to the current molecule a marker, the observer maps the resulting molecule to d, until the strand with a marker is extended or until the molecule is completed.

Suppose that, when the first marker is attached, the length of the strand with that marker is l_1, the length of the other strand is l_2 (clearly, $l_1 > l_2$), and then the output produced so far is $b^{l_1+l_2-2}d$. To complete the molecule by extending the strand without the marker, we need to attach $l_1 - l_2$ symbols to it, and in this case the observer outputs $d^{l_1-l_2-1}\lambda$. Thus, the resulting string x consists of $l_1 + l_2 - 2$ b's and $l_1 - l_2$ d's. Since $l_2 \geq 1$, the difference between the number of b's and the number of d's is $l_1 + l_2 - 2 - (l_1 - l_2) = 2l_2 - 2 \geq 0$. (Recall that in case we attach a symbol to a string with the marker, the observer only outputs λ, so the inequality $m = |x|_b \geq |x|_d = n$ remains valid, and all the combinations (m, n), $m \geq n$ are possible). Hence, $L(\gamma) = L_1$.

6 Small Observable Sticker Systems

The previous example is a preliminary "hint" on how, observing a sticker system, we can get more power with respect to the case when sticker systems are considered in the classical way.

The idea of the previous example can be extended and it is possible to show that there exist observable (simple regular) sticker systems, generating non-context-free languages, even using dominoes of length 1. In other words, the "simple" observation of the evolution of the sticker system permit us to "jump" from a subclass of regular language to non-context-free languages.

Theorem 1. *There exists an observable sticker system $\phi = (\gamma, O)$, $\gamma = (V, \rho, A, D)$, length$(D) = 1$ such that $L(\phi) \notin CF$.*

Proof. Consider the following observable sticker system $\phi = (\gamma, O)$:

$$\gamma = (V = \{a, b, c\}, \rho = \{(a, a), (b, b), (c, c)\}, A = \{\begin{bmatrix}c\\c\end{bmatrix}\}, D),$$

$$D = \{\binom{a}{\lambda}, \binom{\lambda}{a}, \binom{b}{\lambda}, \binom{\lambda}{b}, \binom{\mathbf{c}}{\lambda}, \binom{\lambda}{\mathbf{c}}\}$$

with the observer O defined by the following mapping,

$$H_1 = \begin{bmatrix} c \\ c \end{bmatrix} \begin{pmatrix} U^* a \\ \lambda \end{pmatrix}, \qquad H_2 = \begin{bmatrix} c \\ c \end{bmatrix} \begin{pmatrix} U^* b \\ \lambda \end{pmatrix}, \qquad H_3 = \begin{bmatrix} c \\ c \end{bmatrix} \begin{pmatrix} U^* \mathbf{c} \\ \lambda \end{pmatrix},$$

$$H_4 = \begin{bmatrix} cU^* a \\ cU^* a \end{bmatrix} \begin{pmatrix} U^* \mathbf{c} \\ \lambda \end{pmatrix}, \qquad H_5 = \begin{bmatrix} cU^* b \\ cU^* b \end{bmatrix} \begin{pmatrix} U^* \mathbf{c} \\ \lambda \end{pmatrix}, \qquad H_6 = \begin{bmatrix} cU^* \mathbf{c} \\ cU^* \mathbf{c} \end{bmatrix}$$

$O(w) = a$ if $w \in H_1 \cup H_4$, $O(w) = b$ if $w \in H_2 \cup H_5$, $O(w) = c$ if $w \in H_3 \cup H_6$,
$O(w) = \lambda$, otherwise. , U={a,b}

The language generated by γ is $L_2 = \bigcup_{x \in U^*} \Big(xc \cdot Pref(xc) \cup Pref(x) \cdot Sub(xc) \Big)$. Notice that $L_2 \cap U^* c U^* c = \{xcxc \mid x \in U^*\} \notin CF$, and hence $L_2 \notin CF$.

The computation of the system starts from the axiom $\begin{bmatrix} c \\ c \end{bmatrix}$ (at this point we can consider both strands "empty"), and pieces ("symbols") from D can be adjoined to the strands of the axiom during the computation. When a complete molecule is obtained, the computation stops. To understand the explanation, think of **c** as a marker.

While the marker is not added to the upper strand and the lower strand is "empty" (for molecules of the form H_1 or H_2), the observer outputs, one by one, the symbols added to the upper strand. After some symbol is added to the lower strand, the symbols added to the upper strand are not output anymore (i.e., the observer outputs λ).

As soon as the system adds **c** to the upper strand (for the molecules of the form H_4 or H_5), the observer starts to output the symbols that are adjoined to the lower strand.

If, at some step, a symbol is added to the upper (or lower) strand to the right of the marker **c**, then, starting from such step, the observer will not produce any input anymore.

We can distinguish three main cases in the way the system ϕ works. We can get the string $s = xcxc$ by first adding the symbols of $x\mathbf{c}$ to the upper strand until the marker **c** is adjoined (letting the observer to output xc, symbol by symbol), and then adding the symbols of $x\mathbf{c}$ to the lower strand (letting the observer to output xc again). The observer cannot guarantee that, first the upper strand is completed, and then the lower strand is completed. Therefore, strings different from $xcxc$ can also be generated.

The system ϕ can produce strings in the set $xc \cdot Pref(xc)$ in the following case: suppose the upper strand is completed (obtaining $cx\mathbf{c}$) and the lower strand is being completed; before it finishes, a symbol might be added to the upper strand, at the right of the marker **c**. Starting from this step the observer will output λ until the computation halts.

On the other hand, the system ϕ can also generate strings in the set $Pref(x) \cdot Sub(xc)$. The symbols corresponding to a prefix of x are added to the upper strand (the observer produces $Pref(x)$ as output of this phase). At some step, some symbols (i.e., a prefix of $x\mathbf{c}$) are added to the lower strand, and during this phase the observer outputs λ. At some time the upper strand is completed and **c** is added (during this phase no output is produced because the lower strand is not empty).

Starting from a certain step, new symbols are added to the lower strand, obtaining $cx\mathbf{c}$. Because, during this phase, a symbol might be added at any step to the right of the marker in the upper strand (thus stopping the output), the string produced during this phase is in $Sub(xc)$. Hence, the full output is in the set $Pref(x) \cdot Sub(xc)$. Therefore, the language generated by ϕ is exactly L_2. □

7 Using an Observer with Rejection: Universality

After Theorem 1 it is natural to ask which is the class of sticker systems that is universal when observed by a finite state automaton. Somehow expected, from Theorem 2 to get universality we do not need "complicated" sticker systems but simple regular sticker systems with dominoes of length at most 4 suffice. On the other hand we need to use an observer that is able to discard any "bad" evolution, as the one described in Section 4.

Theorem 2. *For each $L \in RE$ there exists an observable sticker system $\phi = (\gamma, O)$, $\gamma = (V, \rho, A, D)$, with $length(D) \leq 4$ such that $\widehat{L}(\phi) = L$*

Proof. (sketch) For a given language $L \in RE$ there exists a (binary) *conditional* grammar $G = (N, T, P, S)$, generating L. We use the following notations: E, F, Z, M_1, M_2 are new symbols, $U = N \cup T \cup \{E, F, Z, M_1, M_2\}$, $U' = \{X' \mid X \in U\}$, $U'' = \{X'' \mid X \in U\}$, $U_0'' = U'' - \{M_1, M_2, E, F\}$, $U_1 = U \cup U' - \{M_1, M_2, M_1', M_2'\}$, $U_2 = U'' \cup Lab(P) - \{M_1'', M_2''\}$. We associate distinct labels to the productions in P, we write the set of all labels as $Lab(P)$. For every rule $(r : B \to x)$ we define

$$cod(r) = \begin{cases} Z', & \text{if } x = \lambda, \\ C', & \text{if } x = C \in (N \cup T), \\ B_1' M_1' M_2' B_2', & \text{if } x = B_1 B_2 \in (N \cup T)(N \cup T). \end{cases}$$

We construct the following observable sticker system $\phi = (\gamma, O)$:

$$\gamma = (V = U \cup U' \cup U'' \cup Lab(P), \rho, A = \{ \begin{bmatrix} E \\ E'' \end{bmatrix} \begin{pmatrix} M_1 M_2 S M_1 M_2 E \\ \lambda \end{pmatrix} \}, D),$$

$$\rho = \{(X, X''), (X', X''), (X'', X), (X'', X') \mid X \in U\}$$
$$\cup \{(B, r), (r, B), (B', r), (r', B) \mid (r : B \to x) \in P\},$$

$$D = \{ \begin{pmatrix} B \\ \lambda \end{pmatrix}, \begin{pmatrix} B' \\ \lambda \end{pmatrix}, \begin{pmatrix} \lambda \\ B'' \end{pmatrix} \mid B \in U\} \cup \{ \begin{pmatrix} cod(r) \\ \lambda \end{pmatrix}, \begin{pmatrix} \lambda \\ r \end{pmatrix} \mid r \in Lab(P)\}.$$

We consider a morphism $h : U \cup U' \to U$ defined for every $x \in U$ as

$$h(x) = h(x') = \begin{cases} \lambda, & \text{if } x \in \{M_1, M_2, Z\}, \\ x, & \text{otherwise.} \end{cases}$$

To describe the observer, we define a set H of "molecule blocks" (representing the sentential forms of G)

$$H = \{ \begin{bmatrix} x_1E \\ x_2E'' \end{bmatrix} \mid \exists (r : B \to x, R) \in P : h(x_1) \in R,$$
$$x_2 \in (M_1''M_2''(Lab(P) \cup U_0''))^* \cap U''^* r U''^* \},$$

Finally, the mapping of observer O is given below.

$$O(w) = \begin{cases} \lambda, & \text{if } w \in \begin{bmatrix} E \\ E'' \end{bmatrix} H^* \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \begin{pmatrix} x_3Ex_4 \\ \lambda \end{pmatrix} \text{ and} \\ & ((Suff(Ex_4), Suff(Ex_1)) \in U_1 \times \{M_1''\} \\ & \cup \{M_1, M_1'\} \times \{M_1''\} \cup \{M_1, M_1'\} \times \{M_2''\} \\ & \cup \{M_2, M_2'\} \times \{M_2''\} \cup \{M_2, M_2'\} \times U_2 \\ & \cup \{(X, X'') \mid X \in U - \{M_1, M_2\} \\ & \cup \{(cod(r), r) \mid (r : B \to x, R) \in P\}\}, \\ t(X), & \text{if } w \in \begin{bmatrix} E \\ E'' \end{bmatrix} H^* \begin{bmatrix} x_1X \\ x_2 \end{bmatrix} \begin{pmatrix} x_3F \\ \lambda \end{pmatrix} \text{ and} \\ & x_1Xx_2 \in (U \cup U' - N)^*, \\ \lambda, & \text{if } w \in \begin{bmatrix} E \\ E'' \end{bmatrix} H^* \begin{bmatrix} (U \cup U' - N)^*F \\ (U'')^*F'' \end{bmatrix}, \\ \perp, & \text{otherwise,} \end{cases}$$

where $t : U \cup U' \longrightarrow T$ is a morphism defined as $t(X) = \lambda$ if $X \notin T \cup T'$ and $t(X) = t(X') = X$ if $X \in T$. We now proceed to the explanations of the construction above.

Simulating G. γ assembles the molecule representing the concatenation of sentential forms of the derivation in G. Symbol by symbol, we copy the current sentential form, applying some production once (copying or rewriting a symbol takes six steps). O checks the regular condition, rejecting "incorrect" molecules.

Symbols used. In the concatenation of sentential forms, E represents the separator; F marks the final one; Z represents erased symbols. M_1 and M_2 are the spacers, used to synchronize the extension of the upper and lower strands. Elements of N and T are symbols of G.

The union of all the above is denoted by U. In the upper strand, the symbols in U are the ones copied from the previous sentential form and the symbols in U' represent the result of rewriting some non terminal symbols in the previous sentential form. In the lower strand, the symbols determine the future behavior: those in U'' are to be copied, while the ones in $Lab(P)$ represent the production to be applied.

Regular condition. A sentential form x is represented on the upper strand by x (shuffled with Z^*, with some symbols primed), where each symbol is preceded by the spacers M_1 and M_2 (h is a morphism "recovering" x).

We call a "block" a double strand encoding a sentential form, as described above. By H we denote the set of "good blocks": the lower strand contains exactly one symbol from $Lab(P)$ - one rewriting rule is applied; the upper strand satisfies the regular condition, associated to this rule.

The sticker system. Extending the upper strand corresponds to writing a new sentential form, while extending the lower one corresponds to reading the current sentential form. The overhang of the current molecule represents the unread part of the current sentential form together with already produced part of the new one.

To the upper strand we can adjoin the symbols of U (if copied from the previous sentential form), or codes of the right-hand sides of rules in G (defined by the function *cod*). To the lower strand we add the symbols of U'' (for copying) or the labels of the rules (for applying them).

The observer checks a few conditions, rejecting the result of the computation (by writing $\perp$) if they are not satisfied.

The "correct evolution" assumes that the lower strand is extended first, and a strand is never extended twice in a row before F is placed. So, the observer requires that the spacers (from $S_1 = \{M_1, M_1'\}$, from $S_2 = \{M_2, M_2'\}$) and non-spacers (from U_1) alternate in the upper strand (of the form $E(S_1S_2U_1)^*(\{\lambda\} \cup S_1 \cup S_1S_2)$); M_1'', M_2'' and those from U_2, respectively, in the lower strand. If the synchronization fails, then the last symbols of the strands will either be in $S_1 \times U_2$, or in $S_2 \times \{M_1''\}$, or in $U_1 \times \{M_2''\}$, and the result is rejected.

After we "read" a symbol, we either copy it, or rewrite it by some production in G. The observer checks that what we write in the upper strand corresponds to what we read in the lower strand.

It is also the duty of the observer to check the correctness of the blocks: for each non-final sentential form x exactly one rule of G is applied, and that x respects the regular condition, associated to this rule.

When we arrive to the terminal sentential form, F is added to the upper strand. Starting from this step, only the lower strand is extended, and the observer outputs the result by applying the morphism t to the symbols being complemented. □

The following example shows the technique used in the previous theorem.

Example. Consider a conditional grammar $G = (\{S\}, \{a\}, \{(r_1 : S \to aS, Sa^*), (r_2 : S \to \lambda, a^*S)\}, S)$, we illustrate the simulation of the derivation $S \to aS \to a$. The observer outputs the symbol a in step 33, and λ in other cases, halting at step 39.

Step	0							2	4	6				8	10	12
Upper	$\mathbf{E}$	$\mathbf{M_1}$	$\mathbf{M_2}$	$\mathbf{S}$	$\mathbf{M_1}$	$\mathbf{M_2}$	$\mathbf{E}$	M_1	M_2	$\mathbf{a'}$	$\mathbf{M_1'}$	$\mathbf{M_2'}$	$\mathbf{S'}$	M_1	M_2	E
Lower	$\mathbf{E''}$	M_1''	M_2''	r_1	M_1''	M_2''	E''	M_1''	M_2''	a''	M_1''	M_2''	r_2	M_1''	M_2''	E''
Step	0	1	3	5	7	9	11	13	15	17	19	21	23	25	27	29

Step	14	16	18	20	22	24	26	28	30
Upper	M_1	M_2	a	M_1	M_2	$\mathbf{Z'}$	M_1	M_2	F
Lower	M_1''	M_2''	a''	M_1''	M_2''	Z''	M_1''	M_2''	F''
Step	31	32	33	34	35	36	37	38	39

Using theorem 2, the definition of $\widehat{L}(\phi)$ and the fact that recursive languages are closed under intersection with regular languages, we get:

Corollary 1. *There exists an observable sticker system $\phi = (\gamma, O)$, $\gamma = (V, \rho, A, D)$, with length$(D) \leq 4$ such that $L(\phi)$ is a non-recursive language.*

In other words, also if we do not discard any evolution of the observed sticker system then we still get something not recursive; that is really surprising because intuitively the observer's ability to reject bad evolution could seem a powerful and essential feature to get something not "trivial".

8 Concluding Remarks and Research Proposals

In this paper we have presented a new way to look at the generation of languages in the framework of sticker systems. We have applied the system/observer architecture, introduced in [3], to the sticker systems and, in particular, we have introduced the class of observable sticker systems.

In an observable sticker system we have a simple regular sticker system and an observer (i.e., a finite state automaton with singular output) that watches the "evolution" of the molecule, producing as output a label at each step of the computation. We have shown that the combination of a sticker system with an observer can be very powerful even using simple components; in fact, "observing" a *simple regular* sticker system with elementary (i.e., of length 1) dominoes it is possible to obtain non context-free languages (the family of languages generated, in the standard way, by such kind of sticker systems is subregular). If we use a more "clever" observer, able to reject "bad" computations, then we get the universality just using simple regular sticker systems with dominoes of length 4.

Such results have a clear significance for DNA computing. Sticker systems are theoretical models of the annealing operation essentially used in many DNA computing experiments, starting with the pioneering one of Adleman.

Simple regular simple systems - hence of the same kind as those corresponding to the annealing operation from Adleman experiment - generate only regular languages. Observing their evolution by other finite state devices leads, surprisingly, to universality. Informally speaking, "simple" experiments, observed in a clever manner by "simple" tools can thus compute whatever much more complex processes can compute. This is particularly interesting if we think that there exist premises for observing the evolution of a (single) DNA molecule using the already mentioned FRET technique.

Many problems have been left *open*: we have considered a very restricted (and then interesting) kind of observable sticker systems, where the underlying sticker system uses only dominoes of length 1: we do not know what is the lower bound on the generative capacity of such class. Can we get all the regular languages? We conjecture the answer is yes. Can every context-free language be generated?

Due to Corollary 1, we know that, also without rejecting any computation, just observing simple sticker systems with dominoes of length 4, it is possible to get non-recursive languages. Is it possible to get every *RE* language? A positive answer to this question would be quite surprising. Moreover, can the length of the dominoes used in Theorem 2 be decreased? (and then, what is the minimal length to get universality?) We believe that other interesting results can be found in this direction of research.

Acknowledgements. The first author is supported by the project TIC2002-04220-C03-02 of the Research Group on Mathematical Linguistics, Tarragona. The first author acknowledges IST-2001-32008 project "MolCoNet" and also the Moldovan Research and Development Association (MRDA) and the U.S. Civilian Research and Development Foundation (CRDF), Award No. MM2-3034 for providing a challenging and fruitful framework for cooperation.

References

1. L.M. Adleman, Molecular Computation of Solutions to Combinatorial Problems, *Science*, 226, 1994, 1021–1024.
2. A.Alhazov, M.Caveliere, Computing by Observing Bio-Systems: the Case of Sticker Systems. In: C. Ferretti, G. Mauri, C. Zandron (eds.) *Preliminary Proceedings of Tenth International Meeting on DNA Computing (DNA 10)*, University of Milano-Bicocca, Milan, 2004, 324–333.
3. M. Cavaliere, P. Leupold, Evolution and Observation – a New Way to Look at Membrane Systems. In: C. Martìn-Vide, G. Mauri, Gh. Păun, G. Rozenberg, A. Salomaa (eds.) *Membrane Computing*, Lecture Notes in Computer Science 2933, Springer, 2004, 70–88.
4. M. Cavaliere, P. Leupold, Evolution and Observation - A Non-Standard Way to Generate Formal Languages, *Theoretical Computer Science*, accepted.
5. M. Cavaliere, N. Jonoska, (Computing by) Observing Splicing Systems, manuscript, 2004.
6. J. Dassow, Gh. Păun, *Regulated Rewriting in Formal Language Theory*, Springer-Verlag, Berlin, Heidelberg, 1989.
7. R. Freund, Bidirectional Sticker Systems and Representations of RE Languages by Copy Languages, In: Gh. Păun (ed.) *Computing with Bio-Molecules: Theory and Experiments*, Springer-Verlag, Singapore, 1998, 182–199.
8. L. Kari, Gh. Păun, G. Rozenberg, A. Salomaa, S. Yu, DNA computing, sticker systems, and universality, *Acta Informatica*, 35, 5 (1998), 401–420.
9. T. Ha, Single-Molecule Fluorescence Resonance Energy Transfer, *Methods*, 25, 2001, 78–86.
10. J.E. Hopcroft, J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, 1979.
11. Gh. Păun, G. Rozenberg, A. Salomaa, *DNA Computing - New Computing Paradigms*, Springer-Verlag, Berlin, 1998.
12. G. Rozenberg, A. Salomaa, *Watson-Crick Complementarity, Universal Computations and Genetic Engineering*, Technical Report 96-28, Dept. of Computer Science, Leiden University, 1996.
13. A. Salomaa, *Formal Languages*, Academic Press, New York, 1973.

DNA-Based Computation Times

Yuliy Baryshnikov[1], Ed Coffman[2], and Petar Momčilović[3]

[1] Bell Labs, Lucent Technologies, Murray Hill, NJ 07974
[2] Department of Electrical Engineering, Columbia University, New York, NY 10027
[3] IBM T.J. Watson Research Center, Yorktown Heights, NY 10598

Abstract. Speed of computation and power consumption are the two main parameters of conventional computing devices implemented in microelectronic circuits. As performance of such devices approaches physical limits, new computing paradigms are emerging. Two paradigms receiving great attention are quantum and DNA-based molecular computing.

This paper focuses on DNA-based computing. This paradigm can be abstracted to growth models where computational elements called tiles are self-assembled one by one, subject to some simple hierarchical rules, to fill a given template encoding a Boolean formula. While DNA-based computational devices are known to be extremely energy efficient, little is known concerning the fundamental question of computation times. In particular, given a function, we study the time required to determine its value for a given input. In the simplest instance, the analysis has interesting connections with interacting particle systems and variational problems.

1 Introduction

The elementary logic unit of DNA computing is the *tile* modeled as a marked or labeled square. In the simplest version, the label values are 0 or 1, and they break down into two input labels on one edge and two output labels on the opposite edge. A computational step consists of one tile bonding to others according to given rules that match input labels of one tile to the output labels of one or two adjacent tiles.[1] Successive bonding of tiles performs a computation, e.g., the evaluation of a Boolean formula on given inputs, in a *self-assembly* process guided by a *template*. The tiles are DNA-based molecular structures moving randomly, in solution, and capable of functioning independently and in parallel in the self-assembly process; the template is needed to properly structure the self-assembly, in particular to impose the sequential constraints on self-assembly needed to produce the desired computation. In contrast to classical computing paradigms, the random phenomena of self-assembly create a randomness in the time required to perform a given computation. The research reported here characterizes stochastic computing times within computational paradigms based on self assembly.

The notion of computing with tiling systems originated with Wang [1] over 40 years ago, but the modern theory emerged within the last several years in the work of Win-

[1] In the DNA lexicon, the notion of bonding is also commonly referred to as gluing, sticking, or attaching.

free [2], and Rothemund and Winfree [3]. The early theoretical work on general computation focused chiefly on various measures of complexity, in particular, program-size and time complexity [4, 3, 5], but Adleman et al [4, 6] investigated interesting combinatorial questions such as the minimum number of tile types needed for universality, and stochastic optimization questions such as the choice of concentrations that leads to minimum expected assembly times.

The analysis of random phenomena of self assembly has produced few results to date. Adleman et al [7] inaugurated this line of research with an analysis of a baseline linear model of a random self-assembly process. (See [8] for a variant of the linear model.) Baryshnikov et al [9] extended the results to incremental self-assembly systems in which structures grow only one tile at a time, and they introduced a novel control mechanism for maximizing yield in reversible processes. The scale of tile systems in the discrete setting is so large that, to date, the techniques of hydrodynamic limits have shown the most promise. This is further illustrated in the studies of this paper.

To fix ideas, we describe the recently proposed techniques of Carbone and Seeman [10] in Section 2. The next section will then fit this scheme into the abstraction of growth models. Section 3.1 studies computation times as the times to grow rectangular constructs, then Section 3.2 generalizes our timing analysis to the growth of arbitrary structures in two dimensions, and includes a model applicable to error correcting self assembly.

2 DNA Computation of Boolean Formulas

A model for DNA-based computations was proposed in [10], and is illustrated in Figure 1 [10]. The template governing computations is a triangular array of nodes with i nodes at level i, $i \geq 1$. Physically, it may be viewed as an array of posts or nodes, called *pawns*, glued to a metallic substrate. Boolean formulas in the form of trees are mapped onto the template by suitably typing the nodes according to function. Such mappings are not unique in general, but the choice of mapping is of no concern here.

As illustrated in Figure 1, the nodes are of four types: input nodes, where the tiles representing the input are superimposed on the leaves of the tree; gate nodes, where

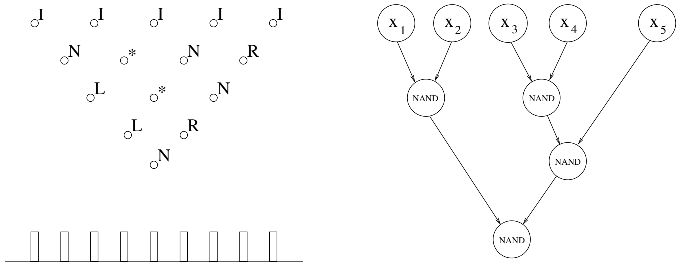


Fig. 1. An example of a template and the Boolean formula it encodes [10]. Under the template of labeled pawns one can see its vertical section. In this example four **N**-tiles are used for computation while ten tiles are needed in total (besides the input)

tiles perform a gating function (the gates in Figure 3 are all NAND gates denoted by **N**[2]; forwarding nodes, where information at a tile input is passed down to the diagonally opposite output; and null nodes, denoted by *, where tiles perform no function at all (these nodes are introduced in order to meet the technical requirement that all nodes of the template be involved in the mapping)

The computation begins by pre-assembling the input tiles to the leaves of the template. Next, the template is put in solution along with the tiles of the various types. The template acts as a lower layer; in an upper layer, tiles then glue to the template by self-assembly; sequential constraints of the hierarchy of gates are ensured by the fact that a tile can bond only if it bonds to tiles at both of its inputs. Each time a gate tile bonds, a Boolean operation is effectively performed. Attachment of the root tile signals completion of the computation and carries the result in its output labels.

3 Growth Models

In a reference theory for self assembly, it is natural to take the times between successive tile placements as independent, exponential random variables; for convenience, we take the mean as the unit of time. As described, the tiling assemblies of the last section are growth processes, although the computations themselves are processes of coalescence which terminate in a single value at the root of the template. As we shall see, it is more useful to focus on the equivalent growth times. In the abstraction introduced below, we relate the times to grow (self-assemble) for constructs or patterns to classical theories of particle processes; growth is again subject to rules analogous to those governing the self-assembly process of the previous section.

In the next subsection we examine a computational template of a rectangular shape. There we outline the basic ideas that will be used in the subsequent section to argue about the speed of computation on templates having arbitrary shapes.

3.1 Rectangle Computation

Figure 2 (left) shows an example in which an initial set of tiles (input) is placed along the two lines (white tiles), and growth proceeds outward in the positive quadrant: the placement of a new tile is allowed only if there are already tiles to the left and below the new tile's position which match labels as before. The left-and-below constraint is equivalent to requiring a newly added tile to bond at both of its input labels. In the example of Figure 2, new tiles can be added only to the three "notches." The eventual output of the computation is represented by the dark tile which can be attached only after all tiles to its left and below are in place. The tiles that perform computation, i.e., non-input and non-output tiles, are lightly colored in Figure 2.

An abstraction of this process operating in the positive lattice is shown on the right in Figure 2. In this model, in which connections to well understood, cognate processes

[2] Recall that a NAND gate with Boolean inputs x and y outputs the value 1 if and only if either x or y or both have the value 0; and recall that any Boolean function can be implemented using only NAND gates.

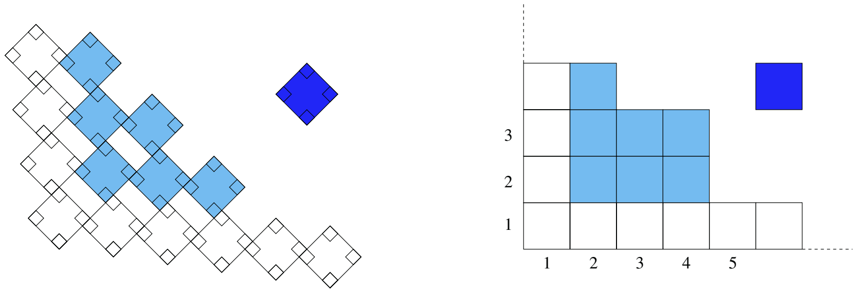


Fig. 2. Growth models of DNA-based computation. The actual process of self assembly (left) can be equivalently represented as a growth process (right). There is a one-to-one correspondence between tiles on the left and squares on the right. White tiles (squares) represent the input

are most easily seen, a new square can be added only in vacant positions having a square both to the immediate left and immediately below the position. Only input squares can lie along the coordinate axes, so under the placement constraint and any finite initial set of squares, a growth process terminates in finite expected time in a rectangular shape. For example, it is easy to see that, after 8 more tiles (squares) are added to the assembly of Figure 2, a rectangular array results (including the (final) output tile). As in the left figure, the result of the ”computation” is depicted by the dark square in the upper-right corner. Clearly, there is a one-to-one mapping between the processes on the left and right in Figure 2. In particular, the template shown in Figure 1 maps to a triangle.

The fundamental quantity of interest is the computation time, or equivalently, the time until the final square (represented by the dark square in position (M,N) of Figure 2) is in place. We denote this random completion time by $C_{M,N}$. Let $T_{i,j}$ be the time it takes for a tile to land at position (i,j) once the conditions are favorable; that is, once both positions $(i,j-1)$ and $(i-1,j)$ are tiled. Recall that our basic assumption is that the $T_{i,j}$'s are independent exponential random variables with unit means. Similarly, by $C_{i,j}$ we denote the time until the square (i,j) becomes occupied. Then the completion times can be written in terms of attachment times $T_{i,j}$ by means of the following recursion

$$C_{i,j} = \max(C_{i-1,j}, C_{i,j-1}) + T_{i,j},$$

with initial conditions $C_{i,1} = 0$ and $C_{1,j} = 0$ for $1 \le i \le M$ and $1 \le j \le N$. The recursion simply reflects the fact that a tile can be attached only if tiles to its left and below are already in place. By unwinding the recursion one can write the completion time as

$$C_{M,N} = \max_{\pi:\ (1,1)\to(M,N)} \sum_{(i,j)\in\pi} T_{i,j},$$

where the maximum is taken over all paths π from $(1,1)$ to (M,N) consisting of segments going north or east only. We remark that the basic recursion formula above can be thought of as an instance of an iterative multiplication of matrices in the so-called *max-plus* algebra, also known as *idempotent algebra* (see [11, 12]). In this formulation, the random values of interest, i.e., the completion times of a computation, are analogous to the matrix elements of the product of a large number of large random matrices.

Alternatively, $C_{M,N}$ is the time when the M-th particle and N-th hole exchange positions in the totally asymmetric simple exclusion process (TASEP) on the integers starting from the megajam configuration: the positions (integers) to the left of the origin are all occupied by particles, and all positions to the right of the origin are empty. At independent, unit-mean exponentially distributed times, particles attempt to move to the right. A move actually occurs only if the adjacent position is a hole, i.e., is unoccupied. We further exploit the correspondence between the TASEP process and the assembly process on the plane in the next subsection.

Remarkably, the exact hydrodynamic-scale behavior of the TASEP process is known. As the rectangular DNA computer becomes large, or equivalently, as N, M grow to infinity such that M/N tends to a positive constant, one has [13–p. 412]

$$\lim_{N\to\infty} \frac{C_{M,N}}{(\sqrt{M}+\sqrt{N})^2} = 1. \tag{1}$$

The preceding formula quantifies the degree of parallelism in the computation. Consider, for simplicity, a square template of size $N \times N$. While the number of tiles is N^2, the computational time is linear in N since $C_{N,N} \approx 4N$ for large N. Expression (1) also allows one to estimate required attachment times given the template size and a constraint on the computation time. For example, let the required computation time be 1 second and let the template consist of $10^3 \times 10^3 = 10^6$ tiles. Then, given that the attachment times are i.i.d. exponential random variables their common mean needs to be $1/(4 \cdot 10^3) = 0.25$ milliseconds.

We conclude this subsection with two observations. First, prior to the computation the set of input tiles needs to be prefabricated. Potentially, the input can be linear in N and, therefore, the prefabrication of the input might be the actual bottleneck of the computation. Second, our analysis assumes that no errors occur in the process of computation, i.e., a tile never bonds to an incorrect place on the template. While the main trade-off in microelectronic circuits is between the speed of computation and power consumption, it appears that in DNA-based computational devices it is between the the speed of computation and the correctness of the output. In particular, higher speed requires smaller attachment times, i.e., higher molecular mobility, which can result in higher error rates.

3.2 Computation of Arbitrary Shapes

In this subsection we examine a DNA computation of an arbitrary shape, say $\lambda\mathcal{D} \subset \mathbb{R}^2$, such that some segments on the boundary of $\mathcal{D}$ correspond to the data, and some to an output, or read-off region (see Figure 4; the input area is the bold line in the southwest, and the output line can be seen in the northeast). We are interested in the case when λ, the scale parameter, is large. In this regime, the number of tiles, and, therefore, elementary logic operations in the DNA computer, is large. As in the previous subsection, the computation is a sequence of attachments of tiles (subject to the left-below condition) to the original template. Before addressing the computational time on such a device, we describe an additional property of the TASEP process.

We focus on an unbounded template (lattice), such as the one in Figure 3, to which tiles can be attached. Consider a profile of the self-assembly process as shown in Fig-

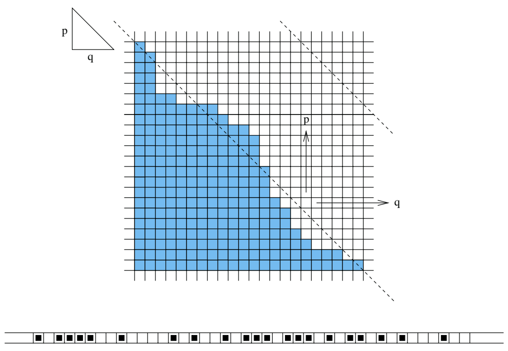


Fig. 3. Boundary of the tiled and untiled regions and its mapping to the particle process. In this particular case $p = q = 1/2$. When the corresponding particle process is in steady state (Bernoulli distribution of particles), the boundary moves to the right at a long-term average speed of q (by symmetry, the vertical speed is p). The angle of the boundary remains the same

ure 3. The profile is the staircase created by the sequence of boundary tiles; these are the dark tiles in Figure 3 that touch, either at a vertex or along a side, the untiled region. Scan the profile from upper left to lower right in a sequence of steps, each being a move left to right across the top of a tile at the boundary or down the right side of a tile at the boundary . In this scan produce a sequence of 0's and 1's, a sample of the TASEP, such that, at each step to the right a 0 is added to the sequence so far, and at each step downward a 1 is added.

Suppose that at some time instant the boundary between the tiled and untiled regions is a random set of tiles with parameter $p \in (0, 1)$. That is, for every tile on the boundary there exists a boundary tile under it with probability p independent of all other tile positions. In this case the corresponding TASEP process has a Bernoulli distribution of particles (ones). Since Bernoulli measure is invariant with respect to TASEP, one concludes that the statistical shape of the boundary remains the same and that the boundary moves to the right at a long-term average speed of q [14–Theorem 4.17] (by symmetry the boundary moves upwards at a long-term average speed of p). Let $v(x, y)$ be the time needed for the boundary to reach point (x, y) (on the hydrodynamic scale). Given that v_x and v_y are partial derivatives of v, the preceding implies that $v_x = q^{-1}$ and $v_y = p^{-1}$ and, thus, since $p + q = 1$, one has

$$v_x^{-1} + v_y^{-1} = 1. \tag{2}$$

Next, in the domain $\mathcal{D}$, consider the function $u(x, y)$, the hydrodynamic limit of the propagation time from the input region on $\partial\mathcal{D}$ to a point (x, y). The claim is that at a point where u is smooth (i.e., where u is continuously differentiable), function u satisfies the following PDE (Hamilton-Jacobi equation):

$$H(u_x, u_y) := u_x^{-1} + u_y^{-1} - 1 = 0, \tag{3}$$

where u_x and u_y are partial derivatives of u. To show that u satisfies this PDE, one could approximate locally level lines of u by random lines described in this section and representing the (random) Bernoulli configuration with an appropriately chosen parameter p to match the local slope of the level curve of u. The propagation speed for such initial configurations is given by (2) and a straightforward coupling argument implies that u satisfies the Hamilton-Jacobi equation locally.

The classic theory, as it relates to variational problems and partial differential equations of first order (see, e.g. [15–Section 9.46]), implies that any function which locally solves $H(u_x, u_y) = 0$ can be represented also as the extremal action function extremizing the following functional

$$\int L\left(\frac{d\xi}{dz}, \frac{d\eta}{dz}\right) dz,$$

where L is the Legendre dual to H, i.e.,

$$L(x,y) = \inf_{(\xi,\eta):\ H(\xi,\eta)=0} (x\xi + y\eta).$$

Evaluating the infimum for the particular choice of $H(\xi,\eta) = \xi^{-1} + \eta^{-1} - 1$ (see (3)), one obtains

$$\begin{aligned} L(x,y) &= x\frac{\sqrt{x}+\sqrt{y}}{\sqrt{x}} + y\frac{\sqrt{x}+\sqrt{y}}{\sqrt{y}} \\ &= (\sqrt{x}+\sqrt{y})^2. \end{aligned}$$

The computation above sketches the following result expressing the computational time in terms of a variational problem:

Theorem 1. *The time $C_{\lambda\mathcal{D}}$ required to complete computation on a DNA computer of shape $\lambda\mathcal{D}$ is given by*

$$\lim_{\lambda\to\infty} \lambda^{-1} C_{\lambda\mathcal{D}} = \sup_{\gamma} \int \left(\sqrt{\frac{d\xi}{dz}} + \sqrt{\frac{d\eta}{dz}}\right)^2 dz, \tag{4}$$

where the supremum is taken over all piece-wise smooth increasing curves $\gamma = (\xi(z), \eta(z)) \in \mathcal{D}$, $0 \le z \le 1$, starting at the input region and ending at the output region of $\mathcal{D}$.

Remark 1. It is easy to prove that the parts of the extremals *in the interior* of $\mathcal{D}$ are straight-line segments; see Figure 4 for examples.

Remark 2. The construction of the function L implies that the functional (4) is parametrization invariant. That is, it depends only on the trajectory $\{\gamma(z)\}_{0\le z\le 1}$.

Remark 3. An essential element of the proof is a *coupling* between the "linear" problem (as represented by the random ragged line on 3) and the original configuration (say, the complement to the first quadrant in Fig. 2). This coupling has the property that the ordering of the configurations by inclusion persists: if $S_1(0) \subset S_2(0)$, then $S_1(t) \subset S_2(t)$ for all $t > 0$.

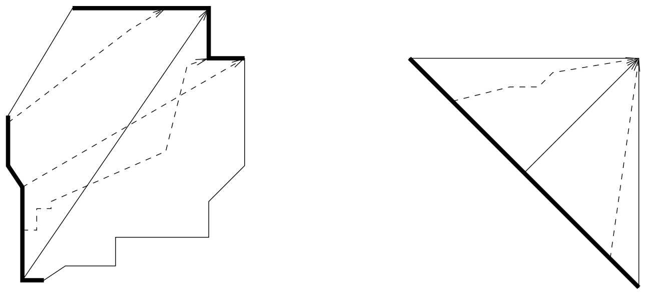


Fig. 4. Two examples of 2D DNA-based computational devices. The one on the right corresponds to the one proposed in [10]. A few piece-wise linear paths from the input areas to the output areas are shown. The paths that determine the computational times (maximize the integral in (4)) are shown with solid lines

In the context of the error-correcting computations, alluded to earlier, the following model plays a fundamental role: it is again a Markov chain with configurations being order ideals on the discrete plane; this time, however, a tile T is allowed to *depart* if it is attached to only 2 other tiles (necessarily, below and to the left of T). Again, we assume independence of all ad- and absorbing events, as long as they are admissible. The rates of departures we assume to be $\rho < 1$. In this model, growth is clearly not monotonic. On the other hand, one can again map this model into the ASEP, the asymmetric simple exclusion process, whose only difference from TASEP is, predictably, the ability of particles to jump to the left (with rate ρ), if the corresponding site is vacant. Again, the coupling mentioned above can be constructed, yielding the following generalization of Theorem 1 (notations are preserved):

Theorem 2. *The time $E_{\lambda\mathcal{D},\rho}$ required to complete computation on a DNA computer of shape $\lambda\mathcal{D}$ with tiles arriving at unit rate and departing at rate ρ is given by*

$$\lim_{\lambda\to\infty} \lambda^{-1} E_{\lambda\mathcal{D},\rho} = \frac{1}{1-\rho} \sup_{\gamma} \int \left(\sqrt{\frac{d\xi}{dz}} + \sqrt{\frac{d\eta}{dz}} \right)^2 dz. \qquad (5)$$

4 Concluding Remarks

This result allows one to determine the hydrodynamic limits of the computation times for 2D DNA-based logical devices. The fluctuations around the hydrodynamic scaling can in fact be estimated. In the particular situation of homogeneous attachment rates, these fluctuations can be studied using combinatorial methods and the powerful machinery of Riemann-Hilbert problems [16, 17]. In more general settings, less precise yet more robust methods are applicable (see, e.g. [18]).

Next we discuss the case when attachment times are not identically distributed exponential random variables, i.e., attachment rates are *nonhomogeneous* and/or *non-Markovian*. In such a case, coupling methods ([19]) allow one to prove once again a convergence of the computation-time function u, in the hydrodynamic scaling, to a solution of a homogeneous first order PDE. However, recovering the exact structure of the PDE appears to be out of reach. Nevertheless, the Markovian setting allows one to bound the corresponding Hamiltonian H if the random attachment times can be bounded by exponential variables. These bounds can also be used, evidently, to put some a priori bounds on the solutions of the PDE. In general, in the hydrodynamic limit, the computation time could be expressed as the extremal value of the following functional

$$\int L\left(\xi, \eta, \frac{d\xi}{dz}, \frac{d\eta}{dz}\right) dz.$$

Although the analytic form of L might not be known, the functional can, in principle, be estimated by simulation. In Figure 5 we show simulation results for two attachment times.

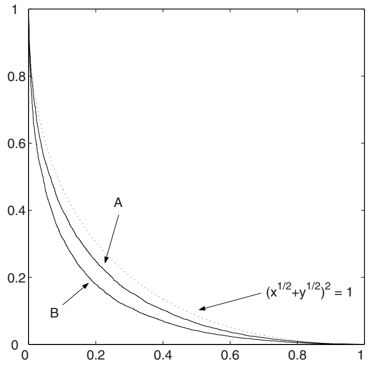


Fig. 5. Estimated forms of the curve $L = 1$ when attachment rates are not exponential random variables. Attachment times are equal in distribution to $[(1+\alpha)1_{\{B=0\}} + (1-\alpha)1_{\{B=1\}}]T$, where B is Bernoulli with parameter 1/2 and T is exponential of unit mean. The parameter α is set to 0.5 (A) and 0.9 (B). The curve $(\sqrt{x} + \sqrt{y})^2 = 1$ shown with the dashed line corresponds to the case of homogeneous attachment times ($\alpha = 0$)

We point out that there is a clear analogy with geometric optics, since we look for the extremals of a Lagrangian that is homogeneous, and of degree-1 in velocities. The space in which the extremals are being sought consists of all (piece-wise linear) curves connecting boundary manifolds. The term *reverse-optic law* serves as a mnemonic for remembering the expression for computation times. In particular, when considering the computation of a *single* bit one has:

DNA computing	$\sup \int \left(\sqrt{\frac{\partial\xi}{\partial z}} + \sqrt{\frac{\partial\eta}{\partial z}}\right)^2 dz$
Geometric optics	$\inf \int \sqrt{\left(\frac{\partial\xi}{\partial z}\right)^2 + \left(\frac{\partial\eta}{\partial z}\right)^2} dz$

Finally, we note that the idea of using *max-plus* algebra in software and hardware design appeared earlier (see [11, 12] and references therein). However, the extension of this circle of ideas to the case of *stochastic* local propagation, crucial for the analysis of DNA computations, is, to our knowledge, new.

References

1. Wang, H.: Proving theorems by pattern recognition, II. Bell System Technical Journal **40** (1961) 1–42
2. Winfree, E.: Algorithmic Self-Assembly of DNA. PhD thesis, California Institute of Technology, Pasadena, CA (1998)
3. Rothemund, P., Winfree, E.: The program-size complexity of self-assembled squares. In: Proc. ACM Symp. Th. Comput. (2001) 459–468
4. Adleman, L., Cheng, Q., Goel, A., Huang, M.D.: Running time and program size for self-assembled squares. In: Proc. ACM Symp. Th. Comput. (2001) 740–748
5. Winfree, E.: Complexity of restricted and unrestricted models of molecular computation. In Lipton, R., Baum, E., eds.: DNA Based Computing. Am. Math. Soc., Providence, RI (1996) 199–219
6. Adleman, L., Cheng, Q., Goel, A., Huang, M.D., Kempe, D., de Espanés, P.M., Rothemund, P.: Combinatorial optimization problems in self-assembly. In: Proc. ACM Symp. Th. Comput., Montreal, Canada (2002) 23–32
7. Adleman, L., Cheng, Q., Goel, A., Huang, M.D., Wasserman, H.: Linear self-assemblies: Equilibria, entropy, and convergence rates. In Elaydi, Ladas, Aulbach, eds.: New progress in difference equations. (Taylor and Francis, London (to appear))
8. Baryshnikov, Y., Coffman, E., Winkler, P.: Linear self-assembly and random disjoint edge selection. Technical Report 03-1000, Electrical Engineering Dept., Columbia University (2004)
9. Baryshnikov, Y., Coffman, E., Momčilović, P.: Incremental self-assembly in the fluid limit. In: Proc. 38th Ann. Conf. Inf. Sys. Sci., Princeton, NJ (2004)
10. Carbone, A., Seeman, N.: Circuits and programmable self-assembling DNA structures. Proc. Natl. Acad. Sci. USA **99** (2002) 12577–12582
11. Litvinov, G., Maslov, V.: The correspondence principle for idempotent calculus and some computer applications. In: Idempotency (Bristol, 1994). Volume 11 of Publ. Newton Inst. Cambridge Univ. Press, Cambridge (1998) 420–443
12. Gunawardena, J.: An introduction to idempotency. In: Idempotency (Bristol, 1994). Volume 11 of Publ. Newton Inst. Cambridge Univ. Press, Cambridge (1998) 1–49
13. Liggett, T.: Interacting Particle Systems. Springer-Verlag, New York (1985)
14. Liggett, T.: Stochastic Interacting Systems: Contact, Voter and Exclusion Processes. Springer, Berlin (1999)
15. Arnol′d, V.: Mathematical methods of classical mechanics. Volume 60 of Graduate Texts in Mathematics. Springer-Verlag, New York (1997) Translated from the 1974 Russian original by K. Vogtmann and A. Weinstein, Corrected reprint of the second (1989) edition.
16. Johansson, K.: Shape fluctuations and random matrices. Comm. Math. Phys. **209** (2000) 437–476
17. O'Connell, N.: Random matrices, non-colliding processes and queues. In: Seminaire de Probabilites XXXVI. Volume 1801 of Lecture Notes in Math. Springer, Berlin (2003) 165–182
18. Talagrand, M.: A new look at independence. Ann. Probab. **23** (1996) 1–37
19. Kozlov, S.M.: The method of averaging and walks in inhomogeneous environments. Russ. Math. Surv. **40** (1985) 73–145 Translation from Usp. Mat. Nauk 40 (1985) 61-120.

Computing Beyond the Turing Limit Using the H Systems

Cezar Câmpeanu[1,⋆] and Andrei Păun[2,⋆]

[1] Department of Computer Science and Information Technology,
University of Prince Edward Island,
Charlottetown, P.E.I., Canada C1A 4P3
`ccampeanu@upei.ca`

[2] Department of Computer Science,
College of Engineering and Science,
Louisiana Tech University, Ruston, P.O. Box 10348,
Louisiana, LA-71272 USA
`apaun@latech.edu`

Abstract. We introduce a new variant of the heavily studied model of H systems. The new variant will use an external factor to determine the set of the active splicing rules. We improve the best known universality result for time-varying H systems with respect to the diameter of such a system and we prove that if the function recording the behavior of the external factor is uncomputable so is the newly defined model, thus exceeding the Turing barrier. We also construct an universal system that is also more powerful than the Turing Machines.

1 Introduction

For more than half a century the Turing machine model of computation was used to define what it means to "compute" or 'to be "computable", notions which are the foundations of the modern theory of computing. In the last few years several researchers have started to look "beyond Turing", i.e., trying to find models of computation that would be able to compute more than a Turing machine (see [1], [3], [4], [15]). This is a very important endeavor, since it means that once such model finds its implementation we would have a computer more powerful than any silicon computer as we know them today. One might argue that this is not possible, and we would like to point out that the speed of our current computers is many times higher than the speed of the biological systems (our brain, as an example), and still we can perform much better/accurate pattern matching than computers.

This observation was the starting point of our work and we will present a model that is capable to compute non-Turing computable languages. The pro-

⋆ The first author is supported by NSERC grant UPEI-600089 and the second author is supported by a LATECH-CenIT grant.

posed model is based on the H systems theory, but it will also take in consideration the environment of the system (an idea borrowed from the P systems in which the environment plays a vital role in the computation). None of the models known so far in the area of H systems try to deal with the computing power over Turing computability, therefore, we think it is of real interest to design a system able to compute languages beyond Turing's limit in this framework.

The reason for doing so is that in real life we can see that many phenomena cannot be explained, mostly because real life systems are not isolated, but they interact with the outside world. Temperature, light, radiation, or simply substance contamination can influence the chemical reactions necessary to recombine DNA strands. An uncontrolled chemical reaction may happen in a normal body when cancerous cells may develop. This kind of reactions are mostly dependent on external factors. For example, to keep a temperature constant is a very difficult task for real life systems, since they need very good insulation. Variation of temperature can speed up or slow down chemical reactions and sometimes temperature can behave like an activator/inhibitor for some reactions. Therefore, it is natural to consider systems where an external factor like temperature can influence DNA splicing.

For an H-system this means we have to add a function τ depending on time, and this function will decide what sets of splicing rules can or cannot be applied at some moment. Hence, the appropriate model for simulating this behavior is a time varying H system where the rules are not applied periodically, but depending on the external conditions, i.e., depending on the value of function τ. Without restricting the generality, we can consider τ as a function from $\mathbb{N}$ to $\{1, \ldots, n\}$, where n is the number of sets of rules that can be activated/deactivated. So, at the moment t, we can consider that only the $\tau(t)$ set of rules can be used.

Hence, we have a new model of τ-time varying H systems. Some natural questions will arise:

1. Can this model be universal? In other words, can we reach the computational power of Turing Machines or other equivalent devices?
2. Is the power of this systems limited to Turing machines? Is the function τ able to add more power to these machines?

In this paper we prove that we can construct an universal system influenced by the temperature and moreover, the computational power will exceed the one of Turing machines in the case that τ is an uncomputable function.

We also improve the best known result for the "usual" time-varying H systems in terms of their diameter/radius of their splicing rules in an effort to bring these systems more closely to an actual implementation. The diameter of the splicing rules is important since the restriction enzymes (that are modeled by a splicing rule) recognize usually small sites on the DNA strand; having arbitrary alphabets in our systems means actually that groups of nucleotides would have to codify one single letter. So, it is clear that there is a significant difference between splicing rules of diameter (3,2,2,2) and (4,5,4,4). Assuming that the alphabet of the system is 10 letters long, then each letter has to be codified with at least 2 nucleotides, making the site recognized by the restriction enzyme modeling the first half of the splicing rule of size 10 for diameter (3,2,2,2) or 18 for diameter (4,5,4,4).

2 Definitions

Let V be an alphabet and $\#, \$$ two symbols not in V. A splicing rule over V is a string of the form $r = u_1\#u_2\$u_3\#u_4$, where $u_1, u_2, u_3, u_4 \in V^*$ (V^* is the free monoid generated by V; the empty string is denoted by λ; for formal language details we refer to [14]).

For $x, y, w, z \in V^*$ and r as above, we write

$$\begin{aligned}(x, y) \vdash_r (w, z) \text{ if and only if } & x = x_1u_1u_2x_2,\ y = y_1u_3u_4y_2,\\ & w = x_1u_1u_4y_2,\ z = y_1u_3u_2x_2,\\ & \text{for some } x_1, x_2, y_1, y_2 \in V^*.\end{aligned}$$

A pair $\sigma = (V, R)$, where V is an alphabet and R is a set of splicing rules, is called an *H scheme.*

For an H scheme $\sigma = (V, R)$ and a language $L \subseteq V^*$, we define:

$$\sigma(L) = \{w \in V^* \mid (x, y) \vdash_r (w, z) \text{ or } (x, y) \vdash_r (z, w), \text{ for some } x, y \in L,\ r \in R\}.$$

A *periodically time-varying H system* (of degree $n, n \geq 1$) is a construct

$$\Gamma = (V, T, A, R_1, R_2, \ldots, R_n),$$

where V is an alphabet, $T \subseteq V$ (terminal alphabet), A is a finite subset of V^* (axioms), and R_i are finite sets of splicing rules over $V, 1 \leq i \leq n$.

Each set $R_i, 1 \leq i \leq n$, is called a *component* of Γ.

At each moment $k = n \cdot j + i, j \geq 0, 1 \leq i \leq n$, the component R_i is used for splicing the currently available strings. Formally, we define

$$L_0 = A,\ \ L_k = \sigma_i(L_{k-1}), \text{ for } i \equiv k (\bmod\ n), k \geq 1, \text{ where } \sigma_i = (V, R_i), 1 \leq i \leq n.$$

The language generated by Γ is defined by $L(\Gamma) = (\bigcup_{k\geq 0} L_k) \cap T^*$.

One of the aims of this paper is to consider a precise estimation of the size of the splicing rules in a time-varying H system, in the sense of [12]. Namely, for a system $\Gamma = (V, T, A, R_1, R_2, \ldots, R_n)$ we define $dia(\Gamma) = (n_1, n_2, n_3, n_4)$, where $n_i = \max\{|u_i| \mid u_1\#u_2\$u_3\#u_4 \in R_j, 1 \leq j \leq n\},\ 1 \leq i \leq 4$. We say that $dia(\Gamma)$ is the *diameter* of Γ.

The family of languages generated by time-varying H systems with at most n components having the diameter less than or equal to $(n_1, n_2, n_3, n_4), n_i \geq 0$, $1 \leq i \leq 4$, is denoted by $TVH_n(n_1, n_2, n_3, n_4)$ (the vector ordering is the natural componentwise one). The family of languages generated by time-varying H systems with at most n components, $n \geq 1$, and of an arbitrary diameter is denoted by TVH_n. By RE we denote the family of recursively enumerable languages (for definitions and properties of the RE sets, see [5]).

As we have already mentioned, we consider an external factor recorded by a function τ that will influence the work of the system. Without loosing the generality, we may assume that this external factor is the temperature. We will consider in the current paper a new variant of the H systems, namely *time-varying H systems with temperature*; we assume that in the system there are n sets of splicing rules, each set being activated/deactivated by the temperature of the environment. To model the temperature activation/deactivation, we will associate with each such system a number from [0,1] written in base n. The number's representation will be $0.n_1n_2n_3...$, where $0 \leq n_1, n_2, n_3, ... \leq n-1$ and will signify that at moment i the active rules are the ones from R_{n_i+1}.

In the following we will study the generating power of this model.

We will consider several cases:

a) the temperature is constant (only one group of rules is continuously activated, e.g., 0.44444444. . .);

b) after finitely many steps the temperature becomes constant (there will be several steps when the temperature varies, but after some moment the temperature remains the same, e.g., 0.1254362154222222222222. . .);

c) the temperature is periodic (e.g., there is a smallest "period" through which the temperature varies, and then becomes as before, e.g., 0.12341234123412. . .);

d) after a while the temperature is periodic,
e.g., 0.437264884729876598765987 65. . . ;

e) the temperature has no period, but it is computable,
e.g., Π–3=0.14159265358979323846264338327950288419716939937510582097. . . .

We will prove now that in the cases a) through e) one cannot go beyond Turing limit. It was proved that $TVH_1 = RE$ in [8], thus there is a TM that will simulate the work of each possible H system. One can construct for each of the cases a) through e) a Turing machine that generates each of those temperatures (each of these cases has the temperature codified as a computable number, so there is for each of them a Turing machine to generate it). One can easily construct now a Turing machine that has as input the codification of the H system and also the codification of the Turing machine generating the temperature and simulate the work of the two machines.
Let us introduce now the interesting case which will be studied in the Section 4:

f) the temperature has no period and it is an uncomputable number;
example: Chaitin's Ω constant (see, for example, [2]).

Since temperature is an unknown variable, τ varying in time it can be represented as a function on $\mathbb{N}$ with values in $\mathbb{N} \cap [0, n-1]$. Therefore, τ behaves like an oracle allowing rules $R_{\tau(t)}$ to be applied at moment t.

The family of languages generated by temperature time-varying H systems with at most n components, where n is greater than one, is denoted by $\tau - TTVH_n$. By $\tau - RE$ we denote the family of τ-recursively enumerable languages [5].

3 Universality Results

In [13] (Theorem 10.8) it is proved that $RE = TVH_n$, for all $n \geq 7$. From the proof, one can see that, in fact, we have $RE = TVH_7(2,3,2,3)$. This was improved in both the number of components and the diameter in [11]: $TVH_n(2,1,1,1) = TVH_n(1,2,1,1) = TVH_n(1,1,2,1) = TVH_n(1,1,1,2) = RE,\ n \geq 4.$. This result is the best one considering only the diameter of the system. Great effort went into diminishing the number of components in such a system and recently it was shown that such H systems with only two components are universal [9], and the current best results (in terms of the number of components) is that one component is enough for universality, [10], [8], [7]. The last two proofs mentioned above have constructions with a diameter of (4,3,4,4) and (4,5,4,4), respectively. We improve here these results by decreasing the diameter of an universal time-varying H system to (3,2,2,2), or (2,3,2,2), or (2,2,3,2), or (2,2,2,3). For our proof we use the universality of type-0 grammars (the previous two proofs were simulating Turing Machines and Tag systems); we think that the grammars are a closer model to the H systems than the Turing Machines and the Tag systems, this being one of the reasons why the diameter could be reduced significantly with respect to the previous constructions. As a secondary note, we would like to point out that most of the universality proofs in this area are simulating grammars, rather than directly the Turing Machines so, other proof techniques could be combined to the current proof if needed.

First, we give an auxiliary result, which will simplify the subsequent investigations.

Lemma 1. *$TVH_n(n_1,n_2,n_3,n_4) = TVH_n(n_3,n_4,n_1,n_2)$, for all $n \geq 1$ and all $n_i \geq 0, 1 \leq i \leq 4$.*

Proof. Consider a time-varying H system $\Gamma = (V,T,A,R_1,\ldots,R_n)$ and construct the system $\Gamma' = (V,T,A,R'_1,\ldots,R'_n)$ with

$$R'_i = \{u_3\#u_4\$u_1\#u_2 \mid u_1\#u_2\$u_3\#u_4 \in R_i\},\ 1 \leq i \leq n.$$

Because $(x,y) \vdash_r (w,z)$ by $r = u_1\#u_2\$u_3\#u_4$ if and only if $(y,x) \vdash_{r'} (z,w)$ by $r' = u_3\#u_4\$u_1\#u_2$, we obtain $L(\Gamma) = L(\Gamma')$. Clearly, if $dia(\Gamma) = (n_1,n_2,n_3,n_4)$, then $dia(\Gamma') = (n_3,n_4,n_1,n_2)$. □

We pass now the main result of this section:

Theorem 1. *$RE = TVH_1(3,2,2,2) = TVH_1(2,2,3,2)$.*
And also $RE = TVH_1(2,3,2,2) = TVH_1(2,2,2,3)$.

Proof. Consider a type-0 grammar $G = (N,T,S,P)$ in Kuroda normal form, that is, with the rules in P of the forms $B \to x, B \to DE, BC \to DE$, for $B,C,D,E \in N$, $x \in T \cup \{\lambda\}$.

Let P_1 be the set of context-free rules in P and P_2 be the set of non-context-free rules in P. We denote the rules in P_1 by $j : u_j \to v_j$, for $1 \leq j \leq m$, and the rules in P_2 by $j : u_j \to v_j$, for $m+1 \le j \le l$. Note that $|u_j| = 1$ for $1 \le j \le m$, and $|u_j| = 2$ for $m+1 \le j \le l$.

We construct the time-varying H system $\Gamma = (V, T, A, R_1)$, with

$$\begin{aligned}
V &= N \cup T \cup \{X, Y, F, Z\} \cup \{X_i, Y_i \mid 1 \le i \le l\},\\
A &= \{XSY, ZF\} \cup \{X_i x Y_i \mid 1 \le i \le m,\ i : B \to x \in P_1, x \in T\}\\
&\quad \cup \{X_i x Y_i \mid 1 \le i \le m,\ i : B \to DE \in P_1, D, E \in N\}\\
&\quad \cup \{X_i DE Y_i \mid m+1 \le i \le l,\ i : BC \to DE \in P_2, D, E \in N\},\\
R_1 &= \{\alpha_1 \# B\alpha_2 \$ X_i \# xY_i,\ \alpha_1 x \# Y_i \$ X_i B \# \alpha_2 \mid i : B \to x \in P_1,\ \alpha_1 \in N \cup T \cup \{X\},\\
&\qquad \alpha_2 \in N \cup T \cup \{Y\},\ B \in N,\ x \in T \cup \{\lambda\}\}\\
&\quad \cup \{\alpha_1 \# B\alpha_2 \$ X_i \# DE,\ \alpha_1 DE \# Y_i \$ X_i B \# \alpha_2 \mid i : B \to DE \in P_1,\\
&\qquad \alpha_1 \in N \cup T \cup \{X\}, \alpha_2 \in N \cup T \cup \{Y\},\ B, D, E \in N\}\\
&\quad \cup \{\alpha_1 \# BC \$ X_i \# DE,\ \alpha_1 DE \# Y_i \$ BC \# \alpha_2 \mid i : BC \to DE \in P_2,\\
&\qquad \alpha_1 \in N \cup T \cup \{X\}, \alpha_2 \in N \cup T \cup \{Y\},\ B, C, D, E \in N\}\\
&\quad \cup \{Z \# F \$ \alpha \# Y,\ \# ZY \$ X \# \beta,\ XZY \# \$ \alpha \# F \mid \alpha, \beta \in T\} \cup \{Z \# F \$ Z \# F\}\\
&\quad \cup \{X_i \# \alpha_1 \$ X_i \# \alpha_1 \mid 1 \le i \le m, i : B \to \alpha_1\alpha_2, \alpha_1 \in N \cup T, \alpha_2 \in N \cup \{\lambda\},\ B \in N\}\\
&\quad \cup \{X_i \# Y_i \$ X_i \# Y_i \mid 1 \le i \le m,\ i : B \to \lambda,\ B \in N\}\\
&\quad \cup \{X_i \# DE \$ X_i \# DE \mid m+1 \le i \le l,\ i : BC \to DE \in P_2,\ B, C, D, E \in N\},
\end{aligned}$$

One can easily see that Γ has the diameter $(3, 2, 2, 2)$. We will prove in the following that the constructed time varying H system Γ has the same language as the grammar G; i.e., $L(\Gamma) = L(G)$.

We first prove that the time-varying H system is capable of generating all the words that are generated by the grammar G; i.e., $L(G) \subseteq L(\Gamma)$. The work of the system is done in two phases: the first phase is simulating the productions from the grammar and the second phase is actually producing the word in the language of $L(\Gamma)$ by removing the special markers from the current word.

We start with a "main" axiom, XSY, in fact we have only the start symbol from the grammar G between two special markers X, *and* Y which mark the start and the end of the word. We will replace S with other nonterminal and/or terminal symbols according to the productions in the grammar G. At some point we choose (nondeterministically) that the current word that appears between X and Y is terminal, which means that by removing from that word the special markers X, Y we generate a word in the language of the H system. This is done by the rules $Z \# F \$ \alpha \# Y$, $\# ZY \$ X \# \beta$, $XZY \# \$ \alpha \# F$ which compose actually the phase two of our simulation, but let us focus on the first phase, the simulation of the grammar productions.

The rules from P_1 are simulated in the following way: let us assume that the current sentential form is XwY, where w is a word over $N \cup T$ and it contains the symbol $B \in N$ for which we have the rule in the grammar G: $k : B \to x$, $x \in T \cup \{\lambda\}$. We have the axiom $X_k x Y_k$ present initially in the system, and because of the rule $X_k \# x \$ X_k \# x$ present in the set of rules it is clear that the aforementioned axiom "survives" through all the steps of the computation, so it is available for splicing with the main word using the following rule: $r_k : \alpha_1\#B\alpha_2\$X_k\#xY_k$. This will produce in one step $(Xw_1|Bw_2Y, X_k|xY_k) \vdash_{r_k} (Xw_1xY_k, X_kBw_2Y)$, where $w = w_1Bw_2$. At the next step in the computation we can apply to these two strings the rule $r'_k : \alpha_1x\#Y_k\$X_kB\#\alpha_2$, which will produce $(Xw_1x|Y_k, X_kB|w_2Y) \vdash_{r'_k} (Xw_1xw_2Y, X_kBY_k)$. At this moment we have correctly simulated the rule $k : B \to x$ from G producing the word Xw_1xw_2Y and the "by-product" X_kBY_k. The simulation of a rule $k : B \to DE$ follows the same path, the only difference is that in the splicing rules the right marker from the axiom (Y_k) is not appearing, in this way the diameter of the system could be kept to a low value.

We are now looking at the way the rules from P_2 $(k : BC \to DE)$ are simulated. Also this process is done in two steps as before, the first splicing is using the rule $r_k : \alpha_1\#BC\$X_k\#DE$ to splice together the main string and the axiom X_kDEY_k: $(Xw_1|BCw_2Y, X_k|DEY_k) \vdash_{r_k} (Xw_1DEY_k, X_kBCw_2Y)$. The second splicing is done according to the rule $r'_k : \alpha_1DE\#Y_k\$BC\#\alpha_2$ and produces the strings Xw_1DEw_2Y and X_kBCY_k. It is easy to see now that all the rules from the grammar G are simulated by the H system in this manner.

The last step of the simulation is to remove the special markers X and Y from the "main DNA strand" in the system. This is done by the rules: $Z\#F\$\alpha\#Y$, $\#ZY\$X\#\beta$, $XZY\#\$\alpha\#F$ which first replace Y by an F. After this, we produce the word ZY, which is able to remove X from the main word and, then, XZY (that is just produced by the last splicing) is able to remove also F from the word. In this way we generate a word in the language of the $L(\Gamma)$ if all the symbols in the main word are terminal at this moment.

We have shown so far the relation $L(G) \subseteq L(\Gamma)$, let us now prove the converse inclusion. We will show that the H system produces no other words than those generated by G. First we look at the rules used to keep the axioms in the system and let them "survive" through all the steps of the computation:

$$\begin{aligned}
&\{Z\#F\$Z\#F\}\\
\cup\ &\{X_i\#\alpha_1\$X_i\#\alpha_1 \mid 1 \le i \le m,\ i{:}B{\to}\alpha_1\alpha_2, \alpha_1 \in N \cup T, \alpha_2 \in N{\cup}\{\lambda\}, B{\in}N\}\\
\cup\ &\{X_i\#Y_i\$X_i\#Y_i \mid 1 \le i \le m,\ i : B \to \lambda,\ B \in N\}\\
\cup\ &\{X_i\#DE\$X_i\#DE \mid m+1 \le i \le l,\ i : BC \to DE \in P_2,\ B, C, D, E \in N\}.
\end{aligned}$$

It is easy to notice that from their specific form all these rules can only be applied to the axioms of the system. This is due to the fact that X_i is always the first letter of a word in the system, and in axioms it precedes the symbol(s) that will replace the nonterminal(s) (according to the rules in G). In all other instances words that X_i appears, it will be followed by the symbol(s) replaced by the rule i from the grammar. This is due the fact that the simulation of such a rewriting rule first cuts after X_k and before the symbol(s) to be rewritten. Following this discussion it is clear now that these rules mentioned above will not produce anything "bad". Another group of rules can be shown that is only leading to terminal configurations only if the rules are applied in the preestablished order: $Z\#F\$\alpha\#Y$, $\#ZY\$X\#\beta$, $XZY\#\$\alpha\#F$. If we replace the Y with a F in the main word, then at the next step we have to use the rule $\#ZY\$X\#\beta$, otherwise the word ZY will not survive to the next configuration of the system, since no other splicing rule can be applied to it, and then X and F will never be removed (they need the words ZY and XZY respectively), which means that, in this case, we will not reach a terminal configuration. One can notice that the removal of X and Y can happen "early" in the simulation, and if we reach a terminal string, then that terminal string will also be reached in the grammar, on the other hand, this could lead to the blocking of the simulation, not leading to any "output", so in this case nothing new can be produced.

We will discuss now the case when the simulation of a rewriting rule from G (that should take two steps for all types of rules from G) is interrupted after the first step and the simulation of yet another rule continues after that. In this case we would have after the first step two words of the form: Xw_1Y_k and X_kw_2Y, k being the rule simulated. At this moment, two more rules could start to be simulated; k' in the first word and k'' in the second word, leading to four words now: $Xw_1'Y_{k'}$, $Y_{k'}w_1''Y_k$, $X_kw_2'Y_{k''}$ and $X_{k''}w_2''Y$. Now,we can continue "breaking-apart" the main word or just finish the simulation of the rules k', k''. If we chose to finish the simulation, then at the next step we would have two words that could finish the simulation of the original rule k and nothing new is produced. One might notice that if only one "half" is simulating a rule and the other part cannot simulate any rules, then that particular string cannot use any other splicing rule, thus disappearing from the system. This will lead to the fact that the simulation of the production cannot complete, thus no terminal configuration will be reached due to the special markers X_k, Y_k present in the string and which cannot be removed from now on. This concludes our justification since we showed that no splicing rule can lead to a terminal configuration that would produce a word not in $L(G)$.

The equality $RE = TVH_1(2, 2, 3, 2)$ follows directly from the Lemma 1.

The second equality mentioned in the theorem: $RE = TVH_1(2, 3, 2, 2)$ requires its own construction. We will give only the basic idea of the construction and leave the details to the reader:

We have to cut after the symbol(s) to be simulated: for a rule $i : BC \to DE$ we would have the splicing rules $BC\#\alpha_1\$DE\#Y_i$ and $\alpha_1\#BCY_i\$X_i\#DE$. The other rules are simulated in a similar way; the other modification to the previous construction is the removing of X, Y which should be performed in a reversed order: first replace X with T and then remove Y, finishing by removing T. The equality $RE = TVH_1(2, 2, 2, 3)$ also follows from Lemma 1. □

4 Computation Beyond the Turing Limit with H Systems

We now proceed to study the power of the new model introduced in this paper: the temperature controlled time-varying H systems. We will start by first noticing that these systems are universal: consider that all the components in a system contain the same set of rules, then the temperature makes no difference in the computation of the system, so they are equivalent in power to the time-varying H systems. Moreover, since we showed in the previous section that the time-varying H systems are universal, it follows that the temperature controlled time-varying H systems are able to generate all RE languages. In the following we show that if the temperature is an uncountable number (τ), then the time-varying H systems controlled by the temperature τ are able to generate non-RE languages.

Theorem 2. *For τ an uncountable number, $\exists \Gamma \in \tau - TTVH_2$ such that $L(\Gamma) \notin RE$.*

Proof. The theorem states that a time-varying H systems with temperature τ is more powerful than Turing Machines if the temperature is an uncountable number. We can consider a time-varying H systems with temperature that has only two components and the temperature is measured with respect to a threshold (if the temperature is below the threshold, the set 1 is activated, if not, the set two is activated). If this number written in base 2 is not calculable, then we will show that there is a τ-H system to compute a language not in RE.

We construct the time-varying τ-H system with temperature $\Gamma = (V, T, A, R_1, R_2)$, with

$$V = \{X, Y, Z, B\} \cup T, \text{ where } T = \{0, 1\}$$
$$A = \{BX, Z0X, Z1X, YY\},$$
$$R_1 = \{B\#X\$Z\#0X,\ \alpha_1\alpha_2\#X\$Z\#0X \mid \alpha_1 \in \{0, 1, B\},\ \alpha_2 \in T\} \cup Q,$$
$$R_2 = \{B\#X\$Z\#1X,\ \alpha_1\alpha_2\#X\$Z\#1X \mid \alpha_1 \in \{0, 1, B\},\ \alpha_2 \in T\} \cup Q,$$

where

$$Q = \{Z0\#X\$Z0\#X,\ Z1\#X\$Z1\#X, Y\#Y\$Y\#Y\} \cup \{B\#\alpha_1\$\#YY \mid \alpha_1 \in T\}.$$

The work of this system is basically to copy the temperature into the generated word; if the temperature activates the set 1, then at that step a "0" is appended to the current string generated by the system if the set 2 was activated, then a "1" is appended. We can also choose nondeterministically to stop the work of the machine by deleting the nonterminal symbol B and, thus, produce a word in $L(\Gamma)$. It is clear now that the language generated is a sequence of words that approximate the temperature by 1, 2, 3... letters, but since the temperature was assumed uncomputable, then also this language is uncomputable, which means that this simple time varying H system generated a language that is not in RE with the help of the temperature. □

In the following we will give a sketch of the construction of a universal time-varying H system that also goes beyond Turing:

Theorem 3. $\tau - TTVH = \tau - RE$

Proof. We will show that we can simulate the work of the oracle Turing machines[1] with such a temperature controlled time-varying H system. The oracle

[1] for more information on oracle Turing Machines and their states $q_?$, q_Y, q_N, and/or $\tau - RE$ we refer the reader to any of the many good Theory of Computation handbooks; a good starting point is also [5].

in the Turing machine is assumed the same as the temperature τ. One can simulate the work of the Turing machine with the time-varying H system, see for example the construction from [8]. The only part of the Turing machine that needs to be considered is the actual oracle and the states that "deal" with the oracle: $q_?$, q_Y, q_N. To do this, one needs to "copy" the temperature into a word in the system (Theorem 2 did exactly this), then read (and erase) the first symbol in the word that "remembers" the temperature and move in the corresponding state q_Y or q_N in the simulated Turing Machine. The temperature could be recorded using the symbols q_Y and q_N and to start the word recording the temperature with an arbitrary number of $q_?$. A sequence of two splicing rules would splice the current configuration word in the Turing machine with the temperature string and replace the $q_?$ with the first symbol in the temperature string, making thus the choice of the oracle. A more delicate matter would require to have the temperature copied in two half strings; the first half being used to simulate the oracle and the second half to just copy the temperature into the string. When the first half is exhausted, one could create a new second half and transform the old second half into a new first half. Due to space restrictions, we will leave the remaining details of the construction to the reader. □

5 Final Remarks

In this paper we prove that real life systems, such as H systems can be more powerful than classical computers in terms of computational complexity, our model being able to simulate Turing machines with oracles. Moreover, the descriptional complexity of our model is the best among all types of time varying H systems.

As future research we would like to continue our investigation for universal H systems with even smaller diameter, and try to apply the idea of the diameter to the time-controlled H systems. It could also be interesting to look at these temperature time-varying H systems as acceptors: we are given an infinite "word" codified as temperature and such a machinery is accepting/rejecting the (temperature)word according to some final state conditions (a state i could be considered final if the number of different DNA strands present in the system is increased (or did not decrease) from the last time that particular set of rules i was applied, etc.). And then, after having such a definition for the final states, one could use different types of accepting methods for infinite words, such as Büchi, Muller, etc.

References

1. C.S. Calude, Gh. Păun, Bio-Steps Beyond Turing, CDMTCS Tech. Rep No 226, 2003, 1–28.
2. G. J. Chaitin, Information, Randomness and Incompleteness, Papers on Algorithmic Information Theory, World Scientific, Singapore, 1987 (2nd ed., 1990).
3. B.J. Copeland, Hypercomputation, *Minds and Machines*, 12, 4 (2002), 461–502.
4. J.-P. Delahaye, La barrière de Turing, *Pour la Science*, 312 October (2003), 90–95.

5. J.E. Hopcroft and J.D. Ullman,*Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, 1979.
6. T. Head, Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviors, *Bull. Math. Biology*, 49 (1987), 737 – 759.
7. M. Margenstern, Y. Rogozhin, Time-Varying Distributed H Systems of Degree 1 Generate All Recursively Enumerable Languages, *Proc. of Workshop on Membrane Computing* (WMC-CdeA 2001), Curtea-de-Argeş, România, 2001, 199–207
8. M. Margenstern, Y. Rogozhin, A Universal Time-Varying Distributed H System of Degree 1, *Proc. of 7th Intern. Meeting on DNA Based Computers* (DNA7) (N. Jonoska, N.C. Seeman, eds.), Tampa, Florida, USA, 2001 and *Lecture Notes in Computer Science* 2340, (N. Jonoska, N.C. Seeman, eds.), Berlin, (2002), 371–380
9. M. Margenstern, Y. Rogozhin, An universal time-varying distributed H system of degree 2, *Preliminary Proc. of Fourth Intern. Meeting on DNA Based Computers*, Pennsylvania Univ., June 1998, 83 – 84.
10. M. Margenstern, Y. Rogozhin, S. Verlan, Time-Varying Distributed H Systems with Parallel Computations: The Problem Is Solved, *Lecture Notes in Computer Science* 2943, (G. Goss, J. Hartmanis, and J. van Leeuwen eds.), Berlin, (2004), 48–53
11. A. Păun, On Time-varying H Systems, *Bulletin of the EATCS*, 67 (1999), 157–164
12. A. Păun, On controlled extended H systems of small radius, *Fundamenta Informaticae*, 31, 2 (1997), 185 – 193.
13. Gh. Păun, G. Rozenberg, A. Salomaa, *DNA Computing. New Computing Paradigms*, Springer-Verlag, Heidelberg, 1998.
14. G. Rozenberg, A. Salomaa, eds., *Handbook of Formal Languages*, Springer-Verlag, Heidelberg, 1997.
15. C. Teuscher, M. Sipper. Hypercomputation: Hyper or computation?, *Communications ACM*, 45, 8 (2002), 23–24.

Biomolecular Implementation of Computing Devices with Unbounded Memory

Matteo Cavaliere[1], Nataša Jonoska[2], Sivan Yogev[3], Ron Piran[4], Ehud Keinan[4,5], and Nadrian C. Seeman[6]

[1] Department of Computer Science and Artificial Intelligence, University of Sevilla, Sevilla, Spain
martew@inwind.it
[2] Department of Mathematics, University of South Florida, Tampa, FL 33620, USA
jonoska@math.usf.edu
[3] Department of Computer Science, Technion, Haifa 32000, Israel
[4] Department of Chemistry, Technion, Haifa 32000, Israel
{sivan_y, ronppy}@techunix.technion.ac.il
[5] The Scripps Research Institute, La Jolla, CA 92037, USA
keinan@scripps.edu
[6] Department of Chemistry, New York University New York, NY 10003, USA
ned.seeman@nyu.edu

Abstract. We propose a new way to implement (general) computing devices with unbounded memory. In particular, we show a procedure to implement automata with unbounded stack memory, push-down automata, using circular DNA molecules and a class IIs restriction enzyme. The proposed ideas are inspired by the results from [1]. The same ideas are extended to show a way to implement push-down automata with two stacks (i.e, universal computing devices) using two circular molecules glued with a DX molecule and a class IIs restriction enzyme. In this case each computational molecule also contains a DX portion. These devices can potentially be incorporated in an array of TX molecules.

1 Introduction

A general idea for using successive restriction cuts on a double stranded DNA in order to simulate a Universal Turing machine was proposed by Rothemund [10]. This was experimentally achieved by Benenson et. al. [1, 2] who have implemented finite state automata with two states and two input symbols. In fact, several different automata were constructed by changing the computational molecules which indicate the state transitions. In [2] they show that the enzyme *FokI* can cleave even if the molecule is nicked which removes the necessity of a ligase.

On the other hand, circular DNA has been proposed for encoding information and using for computing theoretically [4, 9, 13] and implemented experimentally

[5, 8]. In this paper we combine these two ideas and propose using biomolecules (in particular circular DNA strands) and a class IIs restriction enzyme to implement computing devices, push-down automata, with unbounded memory. Since these automata are more powerful than finite state automata, this provides a feasible way for experimental increase of the computational power.

In Section 2 we present a procedure to implement a push-down automaton, that is a computing device with stack as an (unbounded) memory. In classical computation theory (see for ex. [6]), push-down automata are considered the simplest computing devices with unbounded memory and are strictly more powerful than finite state automata. The technique presented here is based on two main ideas: the use of circular DNA strands (which contains the information of the stack and the input symbol) and the use of a unique restriction enzyme, *PsrI*, which is able to cut a DNA strand in two places at the same time. Alternatively, the use of *PsrI* could be substituted with two back to back restriction sites for a class II enzyme similar to *FokI* (similar use was proposed in [10]). The theoretical analysis of splicing systems using this type of enzymes have not been developed and with this paper we hope that such investigations will be initiated.

The potential implementation described here can be considered as a generalization of what was done in [1] where finite state automata were implemented using linear DNA strands. The use of circular DNA molecules allows addition of an unbounded memory to the machine. The basic idea is that the circular DNA contains the instantaneous configuration of the push-down automaton, that means, in a single molecule, the contents of the stack and the content of the input yet to be read are encoded. The enzyme *PsrI* cuts the circular DNA in two places leaving overhangs (sticky ends) corresponding to the elements on the top of the stack-memory (on one side) and to the next input symbol to be read from the input tape (on the other side).

Once the circular DNA has been cut, a "computational" linear DNA strand encoding the transition of the machine that corresponds to the sticky ends is inserted into the circular DNA. In this way, a transition of the push-down automaton is implemented that consists of: an update of the stack memory, a move of the input-head over the input tape and a change of the state. Following the idea used in [1, 2] the state and the input symbols of the machine are encoded as a pair in the sticky end produced by the enzyme cut. To simplify the exposition, we show in detail how to implement a (simple) push-down automaton using two states, two input-symbols and two stack-symbols, accepting the (non regular) language $\{a^n b^n \mid n \in \mathbb{N}\}$.

In Section 3 we show how to implement push-down automata with two stacks using two circular molecules that are attached by two DX molecules and the restriction enzyme *FokI.* The idea is very similar to the one described for implementing push-down automata with one stack. This implementation is particularly interesting because push-down automata with two stacks are equivalent to Turing machines ([6]). We also describe a way how the DX molecule connecting the two "stacks" molecules can be incorporated in a two dimensional array and in that way potentially the whole computational process scaled up.

2 DNA Implementation of Push-Down Automata

2.1 Push-Down Automata with One Stack

In this section, we recall the definition of push-down automata, and some well known theoretical results. This section is mainly based on the material found in the classical automata theory book [6].

A push-down automaton (PDA) is a finite state automaton with a stack memory (called simply, stack). The class of languages recognized (accepted) by PDA's is the class of context-free languages that strictly includes the class of regular languages (recognized by finite state automata).

The PDA has control of both an input tape and a stack (see Figure 1). The stack is the memory of the machine and it works as a "first in - last out" list. That is, symbols may be entered or removed only at the top of the list such that a symbol that is entered (push) at the top pushes the rest of the symbols on the stack one step "down". Similarly, when a symbol is removed (pop) from the top of the list, the remaining symbols on the stack move one step up.

Informally, a transition of a PDA is defined in the following way: at each step, an input-symbol and the stack-symbol at the top of the stack, are read. According to these symbols and the current state, the PDA changes its state and updates the stack, i.e., it either adds a stack-symbol at the top of the stack, removes one stack-symbol from the top of the stack, or leaves the stack unchanged. The computation stops when no transitions can be applied anymore. The input is accepted if (and only if) it has been entirely read and the PDA is in a final state (similarly as in the case of finite state automata).

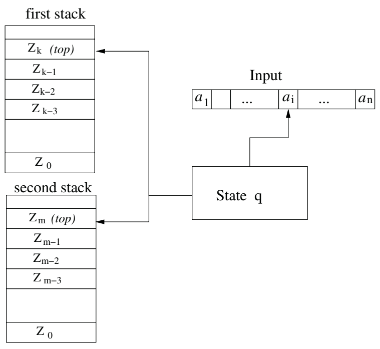


Fig. 1. A Push-down automaton

Well known examples of languages recognized by PDA are the language of *palindrome words* and $\{a^n b^n \mid n \in \mathbb{N}\}$. We give the formal definitions of PDA following [6].

Definition 1. *A push-down automaton M is a system $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ where :*
Q is a finite set of states;
Σ is an input alphabet (its elements are called input-symbols);
Γ is a stack alphabet (its elements are called stack-symbols);
q_0 in Q is the initial state;
Z_0 in Γ is a particular stack-symbol called the start symbol;
$F \subseteq Q$ is the set of final (terminal) states;
δ is the transition mapping from $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma$ to finite subsets of $Q \times \Gamma^*$.

The interpretation of the move (transition):

$$\delta(q, a, Z) = \{(p_1, \gamma_1), (p_2, \gamma_2), \cdots, (p_m, \gamma_m)\},$$

where q and p_i, $(1 \le i \le m)$, are states, a is in Σ, Z is a stack-symbol, and γ_i is in Γ^* is the following. The PDA in state q, reading an input-symbol a with Z as the top stack-symbol, for any i, can enter state p_i, replace symbol Z by string γ_i, and advance the input-head one symbol.

A PDA may also have empty moves

$$\delta(q, \epsilon, Z) = \{(p_1, \gamma_1), (p_2, \gamma_2), \cdots, (p_m, \gamma_m)\}$$

such that the PDA in state q with Z at the top stack-symbol, independently of the input-symbol being scanned, can enter state p_i and replace Z by γ_i for any i, $1 \le i \le m$. We adopt the convention that the leftmost symbol of γ_i will be placed at the top of the stack and the rightmost symbol lowest on the stack.

The PDA stops if a transition from state q, reading input-symbol a_i, and stack-symbol Z is not defined.

If $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ is a PDA, we say $(q, aw, Z\alpha) \rightarrow (p, w, \beta\alpha)$ if $\delta(q, a, Z)$ contains (p, β). We use $\rightarrow^*$ to denote the reflexive and transitive closure of $\rightarrow$. The acceptance of a language by a PDA can be defined in two (equivalent) ways: by entering a final state or by emptying the stack. In this presentation we use the first manner.

Accepted languages (by final states). For a PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ we define $L(M)$ the *language accepted by final state*, to be:

$$\{w \mid (q_0, w, Z_0) \rightarrow^* (p, \epsilon, \gamma), \text{ for some } p \in F \text{ and } \gamma \in \Gamma^*\}.$$

We recall some classical results (see for ex. [6]).

Theorem 1. *The class of languages accepted by PDA's is exactly the class of context-free languages (that strictly includes the class of regular languages).*

Theorem 2. *For every PDA there is an equivalent two state PDA that accepts the same language.*

2.2 Implementing PDA's: An Example

To simplify the implementation of a PDA, without loss of generality, we suppose that every input word is inserted with a symbol indicating end of input denoted with τ. Then a word is accepted by a PDA if, and only if, when reading the end-of-input symbol τ, the PDA stops entering one of the final states. If the PDA stops before reading the end-of-input symbol (i.e. no transitions are possible), then the word is not accepted by the PDA.

In what follows we present a possible implementation of a PDA that accepts the non regular language $L_1 = \{a^n b^n \mid n \in \mathbb{N}\}$. A PDA accepting the language L_1 is the following: $M_1 = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, with $Q = \{0, 1\}$, $\Sigma = \{a, b\}$, $\Gamma = \{Z, \#\}, Z_0 = \#$, $F = \{1\}$.

M_1 has five transitions

(i) $\delta(0, a, \#) = (0, Z\#)$: in state 0, reading an input-symbol a and stack-symbol $\#$, remain in state 0 and add (push) a Z at the top of the stack.
(ii) $\delta(0, a, Z) = (0, ZZ)$: in state 0, reading an input-symbol a and stack-symbol Z, remain in state 0 and push a Z at the top of the stack.
(iii) $\delta(0, b, Z) = (1, \epsilon)$: in state 0, reading an input-symbol b and stack-symbol Z, change to state 1 and remove (pop) a Z from the top of the stack.
(iv) $\delta(1, b, Z) = (1, \epsilon)$: in state 1, reading an input-symbol b and stack-symbol Z, remain in state 1 and pop a Z from the top of the stack.
(v) $\delta(1, \tau, \#) = (1, \#)$: in state 1, reading end-of-input τ, and $\#$ on the top of the stack, remain in state 1 (the computation halts).

In the initial configuration the stack contains only $\#$ and the initial state is $q_0 = 0$; the input-head scans the first symbol of the input string. It is easy to see that the language accepted by M_1 is $L_1 = \{a^n b^n \mid n \in \mathbb{N}\}$. We show how to implement this PDA using the enzyme *PsrI* together with circular molecules containing the information for the stack and the tape, and linear DNA strands for the transitions. The enzyme cleaves as depicted in Figure 2 (for further details see [17]).

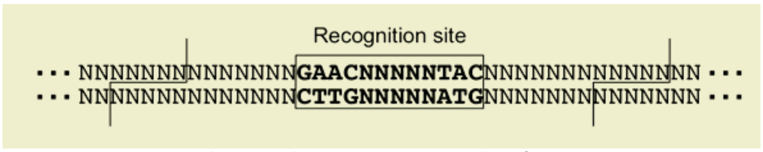

Recognition site
... NNNNNNNNNNNNNNNGAACNNNNNTACNNNNNNNNNNNNNNN ...
... NNNNNNNNNNNNNNNCTTGNNNNNATGNNNNNNNNNNNNNNN ...

Fig. 2. The restriction mode of PsrI

Encoding. The input letters are encoded as $a = TTC$ and $b = AAC$. Codes of the stack symbols, using strings of 5 letters can be chosen $Z = TCCAG$ and $\# = CAAAC$.

The initial circular DNA strand corresponds to the initial configuration of the PDA: it contains the initial configuration of the stack and the input to be read. This is "codified" as follows. The input is written such that any pair of

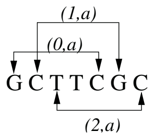


Fig. 3. Coding used to simulate three different states

input-symbols are separated with GC, which is also added in front of the first symbol and after the last symbol. A stop-sequence (the end-of-input) $CAGGC$, follows the input. For example, suppose the input is $aabb$, then it is codified with GCTTCGCTTCGCAACGCAACGCCAGGC. (the separator GC is indicated in italics) The sequence GC allows "moving" between different states (following the idea used in [1]).

This coding allows for *three states* reading the same symbol to be encoded. For example, the symbol a surrounded by GC's is encoded with $GCTTCGC$. We can assume that $GCTTC$ encodes "state 0-reading a", $CTTCG$ encodes "state 1-reading a" and $TTCGC$ encodes "state 2-reading a" (see Figure 3).

Following this idea, in our example we have (we only mention the codes that are used): $GCTTC$ for a in state 0; $GCAAC$ for b in state 0; and $CAACG$ for b in state 1.

The circular DNA strand representing the initial configuration of the PDA is depicted in Figure 6. The first part $CAAAC$ represents the initial configuration of the stack (containing only symbol #; virtually an empty stack). The middle

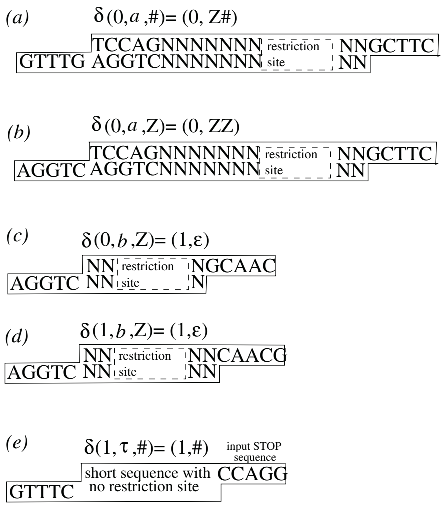


Fig. 4. Transitions of the PDA

portion $GAACNNNNNNTAC$ is the restriction site for the enzyme $PsrI$. The final part is the input as described above.

Transitions. Together with the circular molecule that represents the initial configuration of the PDA five linear DNA strands that encode the transitions of the PDA are also needed: one strand for each transition. The linear strands corresponding to the transitions in our example are depicted in Figure 4. These molecules are added in the solution together with the circular molecules corresponding to the initial configuration of the PDA. The enzyme cuts the circular molecule and the transition molecules are allowed to connect to the circular molecule, after which, the enzyme cleaves again and the process is repeated.

Transitions (a) and (b) in Figure 4 add a symbol Z on the stack, but do not change the states. This is obtained by having NN between the restriction site and the sticky-end representing the input symbol to the right, and after a sequence of seven N's having the sequence $TCCAG$ representing Z on the left hand side. Since the enzyme cleaves seven nucleotides away from the restriction site on both sides, on the right hand side, the next cut will appear at the next input symbol, in same position (i.e. same state) for reading the input sequence, and on the left hand side the symbol 'Z' will be added to the stack. The transitions (c) and (d) are similar, except they do not allow for writing Z's on the stack, but for removing them (the number of N's present between the restriction site and the sticky-end representing the input symbol depends in the way the state-symbol is encoded).

The strand in Figure 4(e) implements the transition 5 and stops in the final state if the end-of-input is read.

In each transition, two ligation reactions occur. Ligation on one side of the transition molecule corresponds to the input, while ligation on the other side corresponds to the stack. It is important to know which side is ligated first, since there is degeneracy in the stack side (it includes only Z or #), and therefore different transition molecules may be ligated at that end at any stage. For example, when the transition molecule in Figure 4(b) is the correct molecule for the next step, the molecules in Figure 4(c) and (d) may be ligated at the stack side, caus-

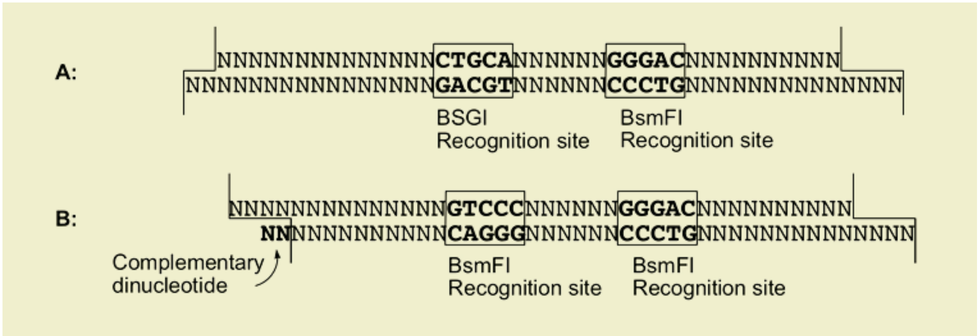

A:
NNNNNNNNNNNNNNCTGCANNNNNNGGGACNNNNNNNNNN
NNNNNNNNNNNNNNNNNGACGTNNNNNNCCCTGNNNNNNNNNNNNNN
BSGI
Recognition site
BsmFI
Recognition site
B:
NNNNNNNNNNNNNNGTCCCNNNNNNGGGACNNNNNNNNNN
NNNNNNNNNNNNNCAGGGNNNNNNCCCTGNNNNNNNNNNNNNN
Complementary
dinucleotide
BsmFI
Recognition site
BsmFI
Recognition site

Fig. 5. Two alternative strategies to enhance the ligation rates at the input end relative to the ligation at the stack end

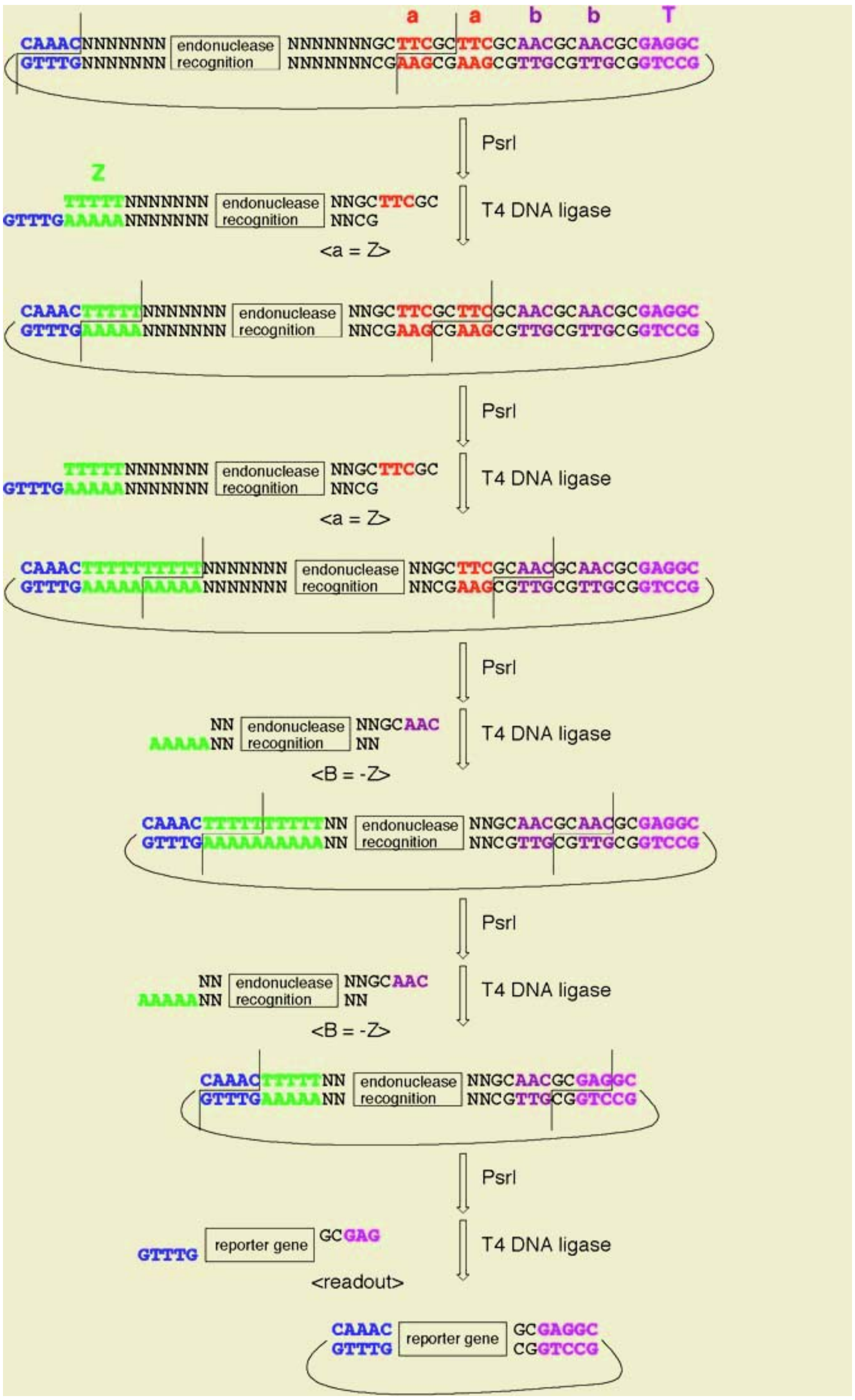

a
a
b
b
T
CAAACNNNNNNN endonuclease recognition NNNNNNNGCTTCGCTTCGCAACGCAACGCGAGGC
GTTTGNNNNNNN NNNNNNNCGAAGCGAAGCGTTGCGTTGCGGTCCG
PsrI
T4 DNA ligase
Z
TTTTTNNNNNNN endonuclease recognition NNGCTTCGC
GTTTGAAAAANNNNNNN NNCG
<a = Z>
CAAACTTTTTNNNNNNN endonuclease recognition NNGCTTCGCTTCGCAACGCAACGCGAGGC
GTTTGAAAAANNNNNNN NNCGAAGCGAAGCGTTGCGTTGCGGTCCG
PsrI
T4 DNA ligase
TTTTTNNNNNNN endonuclease recognition NNGCTTCGC
GTTTGAAAAANNNNNNN NNCG
<a = Z>
CAAACTTTTTTTTTTNNNNNNN endonuclease recognition NNGCTTCGCAACGCAACGCGAGGC
GTTTGAAAAAAAAAANNNNNNN NNCGAAGCGTTGCGTTGCGGTCCG
PsrI
T4 DNA ligase
NN endonuclease recognition NNGCAAC
AAAAANN NN
<B = -Z>
CAAACTTTTTTTTTTNN endonuclease recognition NNGCAACGCAACGCGAGGC
GTTTGAAAAAAAAAANN NNCGTTGCGTTGCGGTCCG
PsrI
T4 DNA ligase
NN endonuclease recognition NNGCAAC
AAAAANN NN
<B = -Z>
CAAACTTTTTNN endonuclease recognition NNGCAACGCGAGGC
GTTTGAAAAANN NNCGTTGCGGTCCG
PsrI
T4 DNA ligase
GTTTG reporter gene GCGAG
<readout>
CAAAC reporter gene GCGAGGC
GTTTG CGGTCCG

Fig. 6. Accepting of the string aabb

ing a halt of the computation process. One way to reduce this problem is to make sure that the first ligation occurs at the input side. Since intermolecular reactions are faster than their intramolecular counterparts, once the ligation occurred at the input side, the probability of correct ligation at the stack side would increase dramatically. This situation may be achieved by reduction of the length of the sticky end at the stack side. Kinetically, annealing processes are faster for longer sticky ends. This goal may be achieved by two alternative ways (see Figure 5):

– Use two different class II restriction enzymes, such as BsgI, which leaves a 2bp sticky end for the stack side, and BsmFI, which leaves a 4bp sticky end for the input side.
– Use two identical restriction sites, such as BsmFI and add a 2bp molecule to the mixture, which will correspond to the inner part of the stack sticky end, causing a need for a 2-step ligation at that end, which will further slow down the ligation rate at that end.

Figure 6 describes the steps of the PDA for accepting the word *aabb*. We start with the initial configuration of the PDA stored in the circular molecule as represented in Figure 6. The consecutive cuts and inserts of the transition molecules are presented in the steps that follow. The cuts end when the 'stop' transition molecule recognizes the end-of-input symbol and it ends in the terminal state. In this case, it has to contain the short sequence without the restriction site.

3 Push-Down Automata with Two Stacks

It is possible to consider push-down automata using two stacks instead of one.

Informally, a transition of a push-down automaton with two stacks (shortly, 2PDA) is defined in the following way: at each step, an input-symbol from the input tape, and the stack-symbols at the top of the first stack and at the top of the second stack, are read. According to the current state, the 2PDA updates both stacks; adds stack-symbols at the top of the stack(s), removes one stack-symbol from the top of the stack(s), or leaves the stack(s) unchanged. At the same time it changes its state and, reads the next input-symbol (as in the case of PDA's, if there is an empty move, the input-head does not advance).

The 2PDA stops if for a given configuration, the next transition is not defined. At start, the 2PDA is in an initial state q_0, the input is placed on the input tape and the first and second stack contain the respective initial stack-symbol Z_0 and Z_1. When the computation stops, the input is accepted if (and only if) it has been entirely read and the 2PDA is in a final state.

The class of languages accepted by 2PDA's is the class of recursively enumerable languages (i.e, 2PDA's are equivalent to Turing machines). The proof of this result and more formal details on 2PDA's can be found in [6].

A graphical description of a 2PDA is described in Figure 7 to the left.

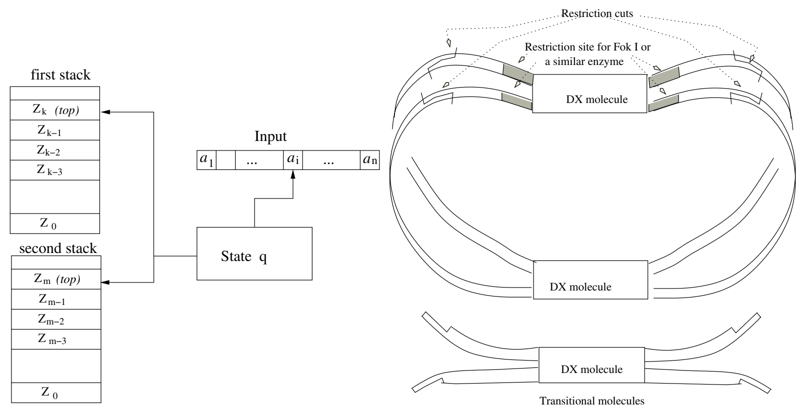


Fig. 7. A graphical description of 2PDA (left) and molecules used for implementation of a 2PDA (right). For simplicity, the middle portion of the body of the DX molecule is not presented

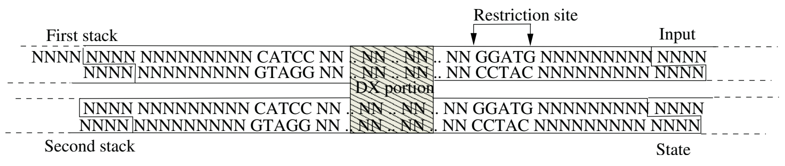

Restriction site
First stack
Input
NNNN NNNN NNNNNNNNN CATCC NN .. NN .. NN .. NN GGATG NNNNNNNNN NNNN
NNNN NNNNNNNNN GTAGG NN .. NN .. NN .. NN CCTAC NNNNNNNNN NNNN
DX portion
NNNN NNNNNNNNN CATCC NN .. NN .. NN .. NN GGATG NNNNNNNNN NNNN
NNNN NNNNNNNNN GTAGG NN .. NN .. NN .. NN CCTAC NNNNNNNNN NNNN
Second stack
State

Fig. 8. An encoding of an instantaneous configuration of a 2PDA in a circular DX molecule

3.1 Implementing 2PDA's Using DX Molecules

In a similar way as presented in Section 2 we can implement 2PDA's i.e., push down automata with two stacks. In this case two circular molecules are "connected" with two DX molecules (called here *circular DX molecule*) and a class IIs restriction enzyme similar to *FokI* (as described in Figure 7, right). The content of the two stacks is stored in the two strands to the "left" of the "upper" DX molecule (the symbols of the two stacks are stored exactly as in the case of the PDA). On the other side of the DX molecule, the symbols of the input string and the current state of the 2PDA are stored, each on one of the strands.

FokI was used in [1] for implementation of a finite state automaton, and we use it in our description for implementing 2PDA. However, any similar enzyme can be used as well.

The restriction site for *FokI* is placed in four places of the strands, connected with the "upper" DX molecule, to be read away from the DX portion of the molecule (see Figure 7 to the right and in a more detail Figure 8 that also depicts the way *FokI* works). As described in Figure 8, the enzyme cleaves the molecule in these four places, and a new computational molecule with sticky ends corresponding to a particular transition of the automaton is inserted. This new molecule has two linear strands with four sticky ends "connected" with a DX portion in the middle. In this way a transition of a 2PDA is simulated.

If the circular DX molecule cannot be cut anymore, or no DX molecule with sticky ends can be inserted in the circular DX molecule, then the computation halts and the circular DX molecule obtained is considered *final*. The final circular DX molecule contains a specific sequence which indicates whether the molecule has been accepted.

The circular DX molecule, containing the initial configuration of a given 2PDA is essentially the same for all 2PDA's. The computation differs only in the set of molecules representing the transitions of the automaton.

Example of one Transition

We present the idea of implementing 2PDA on a simple example simulating a single transition of a 2PDA.

Consider the 2PDA $M_2 = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, Z_1, F)$ with states $Q = \{0, 1\}$, input alphabet $\Sigma = \{a\}$, initial state $q_0 = 0$, alphabet of the stack-symbols $\Gamma = \{Z, \#\}$, final state $F = 1$ and $Z_0 = Z_1 = \#$ are the stack-symbols present at the beginning of the computation in the first and second stack, respectively. Suppose that one of the transitions present in δ is: $\delta(0, a, \#, \#) = (1, Z\#, Z\#)$ meaning, when the 2PDA is in state 0, reads # on both stacks, and a as input-symbol, then, the 2PDA push Z on top of both stacks and changes to state 1.

The symbols and the states used by M_2 can be encoded similarly to the case of PDA M_1 in Section 2. We fix a code composed of 4 letters for the symbols in Σ and Γ. Choose: $a = GTTG$, $Z = TCCA$, and $\# = GCTG$.

Notice that in this implementation the coding of the states is in a "direct" way. There is no need to use the "shift" technique as described in the implementation of M_1. Choose: $0 = TGGT$, and $1 = ACTC$.

The implementation starts with a circular DX molecule corresponding to the initial configuration of the 2PDA M_1. The "upper" part of the circular DX molecule containing the initial configuration is described in Figure 9. Suppose that the input is $a\tau$, where τ is the end-of-input symbol. Set the code of τ to be $CCAG$.

The molecule that corresponds to the transition $\delta(0, a, \#, \#) = (1, Z\#, Z\#)$ is depicted in Figure 10. The idea can be generalized to other transitions; it is very similar to the idea used in the implementation of the transitions of a PDA.

The enzyme *FokI* cuts the circular DX molecule in four places and this can be considered as reading the input-symbol, the current state and the symbols on the top of the two stacks. Once the circular DX molecule is cut, one of the

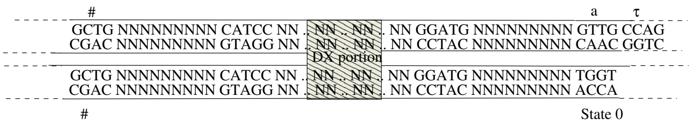

#
a
τ
GCTG NNNNNNNNN CATCC NN .. NN .. NN .. NN GGATG NNNNNNNNN GTTG CCAG
CGAC NNNNNNNNN GTAGG NN .. NN .. NN .. NN CCTAC NNNNNNNNN CAAC GGTC
DX portion
GCTG NNNNNNNNN CATCC NN .. NN .. NN .. NN GGATG NNNNNNNNN TGGT
CGAC NNNNNNNNN GTAGG NN .. NN .. NN .. NN CCTAC NNNNNNNNN ACCA
#
State 0

Fig. 9. Initial configuration of the circular DX molecule

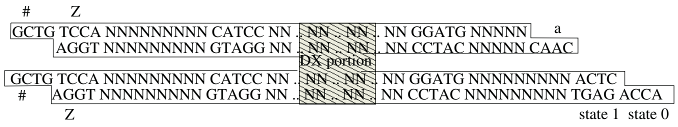

#
Z
GCTG TCCA NNNNNNNNN CATCC NN . NN . NN . NN GGATG NNNNN
a
AGGT NNNNNNNNN GTAGG NN . NN . NN . NN CCTAC NNNNN CAAC
DX portion
GCTG TCCA NNNNNNNNN CATCC NN . NN . NN . NN GGATG NNNNNNNNN ACTC
#
AGGT NNNNNNNNN GTAGG NN . NN . NN . NN CCTAC NNNNNNNNN TGAG ACCA
Z
state 1
state 0

Fig. 10. DX molecule for the transition: $\delta(0, a, \#, \#) = (1, Z\#, Z\#)$

molecules representing the transitions is inserted into the circular DX molecule. This "insertion" simulates the application of a transition of the 2PDA.

The transition $\delta(0, a, \#, \#) = (1, Z\#, Z\#)$ is applied to the initial configuration of the 2PDA, M_2, by having all four sticky ends connected with a corresponding transition molecule. In our example, the DX molecule inserted is the one represented in Figure 10, and the new configuration obtained is such that the next four cuts will correspond to "reading" the end-of-input symbol τ and stack-symbol Z on both stacks and "being" in state 1.

Different transitions can be implemented introducing different transition molecules (with different sticky ends). As in the case of PDA's, the sticky ends can be adjusted to simulate both push and pop over the stacks, and also the empty moves. Similarly, when no cuts can be done on the circular DX molecule or no transition molecule with corresponding sticky ends can be inserted then the circular DX molecule obtained is final. Checking the presence of some special terminal sequence in the final DX molecule determines whether the input is accepted or not.

3.2 2PDA in an Array

Double and triple cross-over molecules have been used as tiles and building blocks for large nanoscale arrays [7, 14, 15]. The assembly of such two-dimensional array can be used to incorporate the circular DX molecules of a 2PDA. The body of the "bottom" DX portion of the circular DX molecule used for storing the instantaneous configuration of a 2PDA can be incorporated in a triple cross-over molecule, such that the sticky ends of the third duplex, from each side, diagonally opposite, are used for connecting the molecule in a TX based array. Schematically this is presented in Figure 11. The drawing aims to give an impression of the means by which the 2PDA units could be inserted into the array; we recognize that it will be much larger than shown, and the C tiles attached to the 2PDA units will need to be much less densely packed.

By including the 2PDA molecules in an array we can potentially (a) scale up the computational process and (b) avoid mutual interactions between the circular DX molecules and formation of dimers and faulty computation that could lead to a wrong result. This process, however, in order to determine precise positions of the 2PDA molecules will require careful coding of the two dimensional array similarly as was done in the bar-code design [16] or in the case of the design of the Sierpinski triangle [11].

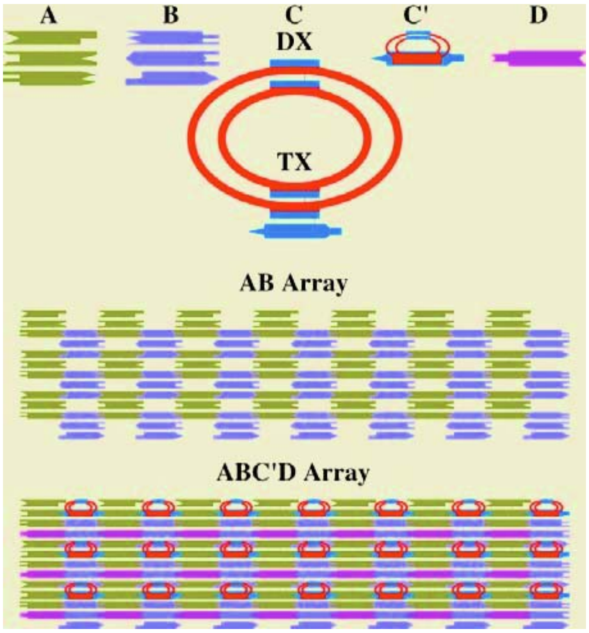


Fig. 11. Insertion of circular DX molecule storing instantaneous configuration of a 2PDA in an array. C is drawn in the same plane as TX molecules A and B, but C' has been rotated nearly perpendicular to the AB array. D is linear duplex filler

4 Concluding Remarks

We have presented a possible way to implement push-down automata (i.e., finite state automata with unbounded stack memory), using a class IIs restriction enzyme, circular and linear DNA strands. In the hierarchy of classical computing devices, push-down automata are the simplest computing devices using unbounded memory and are strictly more powerful than finite state automata.

In particular, an implementation of a small PDA, with two states, two input-symbols, and two stack-symbols, accepting the non regular language $\{a^nb^n \mid n \in \mathbb{N}\}$ was shown. Using the same idea the implementation of a PDA that checks if a string is palindrome is straight forward. Palindromes are a standard example of non regular languages that may be of interest from biological point of view.

From the theoretical point of view, it is easy to describe implementation of a general (non deterministic) PDA with two states and two input-symbols, even using empty-moves in the same manner. Due to Theorem 2 this is enough to implement every PDA, but, the number of stack-symbols may increase. In the implementation presented here, the number of possible stack-symbols is limited to 4^5 as the code of each symbol uses 5 nucleotides. This bound is even smaller since other coding constrains may apply. There are methods to reduce the number of stack symbols ([3]) but in this case the number of states increases.

Therefore it is significant to determine how many stack-symbols can be really codified for a DNA implementation using one enzyme $PsrI$ or $FokI$.

The paper also includes a way to implement push-down automata with two stacks. This implementation uses again a class IIs restriction enzyme and circular DX molecules. This result is of particular interest because such class of computing devices is equivalent to Turing machines. Although we used an identical coding for both stacks in our example, this should be avoided such that no cross annealing of the stacks and the transition molecule occurs. It can be adjusted similarly as in the case of PDA when three symbol alphabet is used for the symbols followed by a GC or AT indicating the two different stacks.

Incorporating circular DX molecules in an array may be a challenging task experimentally, in particular, encoding and assembling the two-dimensional array such that the sticky ends for annealing with the 2PDA molecule appear in the right positions.

Acknowledgment

Parts of this research was performed while the first author was visiting University of South Florida during the Fall of 2003.

The first author is supported by a grant from the Spanish Ministry of Culture, Education and Sport under the Programa Nacional de Formación de Profesorado Universitario (FPU). Seeman has been supported by grants GM-29554 from the National Institute of General Medical Sciences, N00014-98-1-0093 from the Office of Naval Research, grants DMI-0210844, EIA-0086015 (also supporting Jonoska), DMR-01138790, and CTS-0103002 from the National Science Foundation. Keinan is supported by the German-Israeli Project Cooperation (DIP) and by the Skaggs Institute for Chemical Biology.

References

1. Y. Benenson, T. Paz-Elizur, R. Adar, Eh. Keinan, Z. Livneh, Eh. Shapiro, *Programmable and autonomous computing machine made of biomolecules*, Nature, **414** (2001), 430-434.
2. Y. Benenson, R. Adar, T. Paz-Elizur, Z. Livneh, Eh. Shapiro, *DNA molecule provides a computing machine with both data and fuel*, Proc. Nat. Acad. Sci. (PNAS) **100** 5 (2003), 2191-2196.
3. J. Goldstine, J.K. Price, D. Wotschke, *On reducing the number of stack symbols in a PDA*, Math. Systems Theory **26** 4 (1993), 313-326.
4. T. Head, *Splicing schemes and DNA*, in Lindenmayer Systems: Impact on Theoretical Computer Science and Developmental Biology (G. Rozenberg, A. Salomaa, eds.), Springer, Berlin, 1992, 371-383.
5. T. Head et.al, *Computing with DNA by operating on plasmids*, BioSystems **57** (2000), 87-93.
6. J.E. Hopcroft, J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, 1979.

7. T. LaBean, H. Yan, J. Kopatsch, F. Liu, E. Winfree, J.H. Reif, N.C. Seeman, *The construction, analysis, ligation and self-assembly of DNA triple crossover complexes*, *J. Am. Chem. Soc.* **122** (2000), 1848-1860.
8. Liu Q. et al. *DNA computing on surfaces*, *Nature* **403** (2000), 175-179.
9. Gh. Păun, G. Rozenberg, A. Salomaa, *DNA Computing - New Computing Paradigms*, Springer-Verlag, Berlin, 1998.
10. P.W.K. Rothemund, *A DNA and restriction enzyme implementation of Turing machines*, *DNA Based Computers* AMS DIMACS series **27** (1998), 75-119.
11. P. Rothemund, N. Papadakis, E. Winfree, *Algorithmic Self-assembly of DNA Sierpinski Triangles*, *Preproceedings of 9th DNA Based Computers*, Madison, Wisconsin June 1-4 (2003), 125.
12. J. Sakamoto et al., *State transitions by molecules*, *Biosystems* **52** (1999), 81-91.
13. Y. Sakakibara, H. Imai, *A DNA-based Computational Model Using a Specific Type of Restriction Enzyme*, *Proceedings of the 8th DNA Based Computer*, (M. Hagiya, A. Ohuchi eds.), LNCS 2568, 315-325.
14. E. Winfree, F. Liu, L.A. Wenzler, N.C. Seeman, *Design and self-assembly of two-dimensional DNA crystals*, *Nature* **394** (1998), 539-544.
15. E. Winfree, X. Yang, N.C. Seeman, *Universal computation via self-assembly of DNA: some theory and experiments*, *DNA computers II*, (L. Landweber, E. Baum eds.), AMS DIMACS series **44** (1998), 191-214.
16. H. Yan, T.H. LaBean, L. Feng, J.H. Reif, *Directed Nucleation Assembly of Barcode Patterned DNA Lattices*, *Proc. Nat. Acad. Sci. (PNAS)* **100** No. 14 (2003), 8103-8108.
17. *http://rebase.neb.com/rebase/rebase.html.*

Characterization of Non-crosshybridizing DNA Oligonucleotides Manufactured *In Vitro*

Junghuei Chen[1], Russell Deaton[2], Max H. Garzon[3], Jin Woo Kim[4], David Wood[5], Hong Bi[1], Dylan Carpenter[4], and Yu-Zhen Wang[1]

[1] Chemistry and Biochemistry, University of Delaware, Newark, DE, USA 19716
junghuei@udel.edu

[2] Computer Science and Engineering, University of Arkansas, Fayetteville, AR, USA 72701
rdeaton@uark.edu

[3] Computer Science, University of Memphis, Memphis, TN USA 38138
mgarzon@memphis.edu

[4] Biological Engineering, University of Arkansas, Fayetteville, AR, USA 72701
jwkim@uark.edu

[5] Computer and Information Sciences, University of Delaware, Newark, DE, USA 19716
wood@mail.eecis.udel.edu

Abstract. Libraries of DNA oligonucleotides manufactured by an *In Vitro* selection protocol were characterized for their non-crosshybridizing properties. Cloning and sequencing after several iterations of the protocol showed that the sequences, in general, became more non-crosshybridizing. Gel electrophoresis of protocol product, also, indicated non-crosshybridization, and showed evolution in the population of molecules under the non-crosshybridization selection pressure. Melting curves of protocol product also indicated non-crosshybridization when compared to control samples. Thus, it appears that the protocol does select populations of non-crosshybridizing sequences.

1 Introduction

The template-matching hybridization reaction between DNA oligonucleotides is an increasingly important technology, not only for biological applications, such as disease detection with DNA microarrays[1], but also for computations with DNA[2, 3] and formation of DNA-based nanostructures[4, 5, 6]. In solution, single-stranded DNAs randomly diffuse, and when a sufficiently complementary sequence is encountered, a double-stranded molecule (duplex), in which some fraction of base pairs are Watson-Crick complements, is formed. Thus, thermodynamically stable duplexes can form that are not perfect Watson-Crick complements. The difficulty is that DNA computations and nanostructures are designed to form via hybridizations between perfect Watson-Crick complements.

The DNA word design problem is to design DNA oligonucleotide sequences that hybridize as planned and minimize problematic, non-Watson-Crick crosshy-

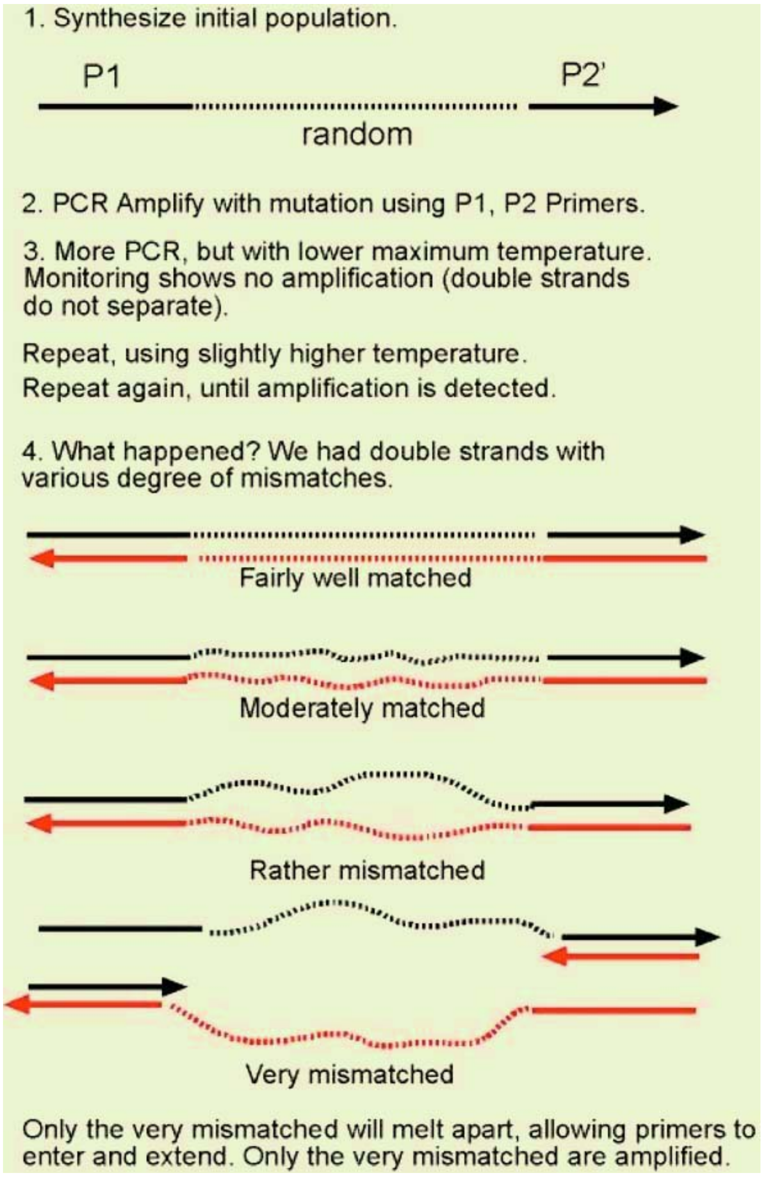

1. Synthesize initial population.
P1
P2'
random
2. PCR Amplify with mutation using P1, P2 Primers.
3. More PCR, but with lower maximum temperature. Monitoring shows no amplification (double strands do not separate).
Repeat, using slightly higher temperature.
Repeat again, until amplification is detected.
4. What happened? We had double strands with various degree of mismatches.
Fairly well matched
Moderately matched
Rather mismatched
Very mismatched
Only the very mismatched will melt apart, allowing primers to enter and extend. Only the very mismatched are amplified.

Fig. 1. Protocol to select maximally mismatched oligonucleotides

bridizations. Solution of this problem is difficult[7], and computationally expensive. Many approaches using digital computers and software have been tried (for a recent review, see [8]), and several designs have been experimentally confirmed[9, 10], though the library sizes have been relatively small. All computer-aided design methods suffer an inability to scale the size of the designed sets, and to capture enough of the chemistry to make the results realistic without long design times.

In response to the limitations of computer-based techniques, a protocol was developed to manufacture non-crosshybridizing libraries of oligonucleotides *in vitro* (Figure 1)[11]. The potential advantages are that the oligonucleotides are selected in the conditions under which computations will be done, and the potential for very large libraries. The starting population of molecules consist of interior random regions to which designed sequences are attached at either end. The protocol works by controlling reaction conditions such that maximally mismatched oligonucleotides melt, while better matched ones do not. Then, using the specified end sequences as primers, the maximally mismatched oligonucleotides are copied by polymerase. Thus, the non-crosshybridizing oligonucleotides are selected through the relative thermodynamic stability of the duplexes that they form. By repeating this process, the percentage of non-crosshybridizing oligonucleotides in the population should increase over time. In previous work[11], it was shown experimentally that the protocol actually selected for maximally mismatched pairs. In addition, simulations of the protocol[12] suggest that selection of noncrosshybridizing sequences occurs over a relatively few rounds of the protocol.

Subsequent to verifying that the protocol works at a basic level, the next challenge was to characterize the libraries produced by the protocol. This requires that their non-crosshybridizing properties be confirmed, and that the number of distinct sequences in the library be estimated. In this paper, the non-crosshybridizing characteristics of the manufactured libraries are investigated experimentally. First, protocol product was cloned and sequenced in order to test for non-crosshybridization. Protocol product was analyzed using gels to determine the overall degree of hybridization in the libraries, and the progress of the selection. Finally, spectroscopic techniques are used to check for crosshybridization. In summary, iteration of the protocol seems to produce sets of non-crosshybridizing sequences. Further work is needed to determine the exact number is unknown.

2 Cloning and Sequencing of Library Oligonucleotides

Starting material, and product from 1 to 4 cycles of the protocol were cloned and sequenced. These sequences were then analyzed using the nearest-neighbor model of duplex thermal stability[13], and a software design tool implemented by the authors[7]. Sequences designed with this tool have been experimentally

Table 1. Sample of Sequences cloned at various iterations of the protocol

Starting Material	
0	GTCATCAGAAAACCACTTTC
1	CCCCATATAGTCTCACGTAC
Cycle 1	
2	TAGGTTCTTTAAGTAACGAT
3	AAAAAATAAGGCTGACCATC
4	TGCTAGAGCGCTATGTTAAG
5	CGTCCCAGCAAACACGCCTG
Cycle 2	
6	TTCCGATCAGATAAGAACTT
7	ATGAAAGCCTGCCTTGCACG
8	TCATGCGATAACGAAACGAC
9	TTCTTCATGCAGAACGCAAC
Cycle 3	
10	CAGACCTGGGTTCGGCCTTA
11	ATAAATTAATGTATCCTAAA
12	CCCAGATAGTGTCAAGTAGG
Cycle 4	
13	TTCCGATCAGATAAGAACTT
14	GTGAGACCATACAGTTAACT
15	TACGGCTCTATGGGAAGCAG

verified to be non-crosshybridizing[9]. The sequences are given in Table 1, and the results of the analysis are shown in Tables 2.

Though the sample of sequences was too small to state any definitive conclusions, the purpose was to gage the overall ability of the protocol to select non-crosshybridizing oligonucleotides, and to eliminate pathological results, such as one or relatively few sequences being selected, or total lack of non-crosshybridization. Because theoretically the starting population contains every possible 20-mer, the chances of picking two sequences at random that crosshybridize is very small. Nevertheless, as the selection protocol was iterated, the non-crosshybridizing properties of the sequences seemed to improve. For example, after cycle 1, there were two crosshybridizations, sequence 2 to sequences 7 (2-7) and 5-7, and after cycle 2, there were three (1-7, 3-7, and 4-7). In cycles 3 and 4, there were no crosshybridizations. All cycles had examples of self-hybridization (hairpins). The protocol does not explicitly select out self-hybridization. Most of the crosshybridizations, including the self-hybridizations, would have been marginally stable at the selection temperature of 43°C. The exceptions are for cycle 1 (4-4 and 5-5), and for cycle 2 (1-1, 4-6, 6-6, and 7-7). Thus, based upon this sample, the protocol seems to be accomplishing its purpose of selecting populations that are progressively more non-crosshybridizing.

Table 2. Pairwise comparison of energetics of cloned sequences. Melting temperature (T_m and minimum Free Energy of Formation ΔG calculate at 1 μM DNA concentration and 1 M NaCl concentration using nearest-neighbor model of duplex thermal stability. Melting temperatures that are less than zero indicate no crosshybridization. Energies and melting temperatures of Watson-Crick pairs (consecutive numbers starting with even) are included for comparison

Seq 1	Seq 2	T_m (°C)	ΔG (kcal/mole)
Starting Material			
0	0	-42.30	-1.98
0	1	65.50	-27.34
0	2	-72.58	-1.10
0	3	-17.91	-2.74
1	1	-42.30	-1.98
1	2	-16.71	-2.75
1	3	-72.58	-1.10
2	2	-22.61	-3.77
2	3	67.48	-28.05
3	3	-22.61	-3.77
Cycle 1			
0	0	-3.40	-3.37
0	1	61.08	-24.57
0	2	-66.07	-1.77
0	3	-48.93	-2.32
0	4	-14.94	-1.84
0	5	-24.87	-2.79
0	6	-48.93	-2.32
0	7	-49.13	-2.50
1	1	-14.69	-2.75
1	2	-70.05	-1.59
1	3	-47.86	-2.06
1	4	-24.87	-2.79
1	5	-45.58	-1.69
1	6	-53.48	-2.20
1	7	-53.48	-2.20
2	2	-24.42	-3.50
2	3	64.73	-26.85
2	4	-59.28	-2.05
2	5	-59.28	-2.05
2	6	-22.64	-4.11
2	7	8.63	-5.99
3	3	-92.43	-0.93
3	4	-59.28	-2.05
3	5	-59.28	-2.05
3	6	-65.59	-1.78
3	7	-22.64	-4.11
4	4	21.66	-8.46
4	5	67.60	-28.60
4	6	-25.68	-3.64
4	7	-31.52	-3.56
5	5	21.66	-8.46

Seq 1	Seq 2	T_m (°C)	ΔG (kcal/mole)
Cycle 1			
5	6	-41.37	-3.14
5	7	0.33	-4.02
6	6	-53.48	-2.20
6	7	78.09	-33.68
7	7	0.73	-5.19
Cycle 2			
0	0	-24.20	-2.53
0	1	63.83	-26.59
0	2	-67.40	-1.17
0	3	-72.14	-1.11
0	4	-99.82	-0.926
0	5	-1.18	-3.80
0	6	-18.66	-3.52
0	7	-15.08	-3.96
1	1	16.19	-7.34
1	2	-55.81	-1.14
1	3	-66.03	-1.56
1	4	-11.89	-4.46
1	5	-69.93	-1.28
1	6	-15.08	-3.96
1	7	12.94	-5.42
2	2	-23.07	-3.83
2	3	74.85	-32.00
2	4	-40.35	-2.27
2	5	-34.30	-2.25
2	6	-23.07	-3.83
2	7	-23.07	-3.83
3	3	21.55	-8.21
3	4	-50.78	-2.27
3	5	-40.35	-2.27
3	6	-23.07	-3.83
3	7	9.56	-5.98
4	4	-22.56	-2.83
4	5	68.90	-29.73
4	6	-13.23	-4.73
4	7	30.62	-10.73
5	5	-33.97	-2.87
5	6	29.93	-10.27
5	7	-13.23	-4.73
6	6	21.81	-8.26
6	7	69.73	-29.82
7	7	30.75	-10.24

Table 2. *Continued*

Seq 1	Seq 2	T_m (°C)	ΔG (kcal/mole)
Cycle 3			
0	0	2.94	-5.80
0	1	74.47	-31.29
0	2	-131.09	-0.19
0	3	-70.05	-1.59
0	4	-4.84	-5.39
0	5	-35.90	-2.69
1	1	-11.30	-4.85
1	2	-70.05	-1.59
1	3	-63.28	-0.90
1	4	-34.79	-2.84
1	5	-4.84	-5.39
2	2	9.77	-5.67
2	3	54.40	-22.04
2	4	-41.03	-2.44
2	5	-68.23	-1.29
3	3	-0.93	-4.02
3	4	-42.85	-1.44
3	5	-70.05	-1.59
4	4	-42.01	-0.92
4	5	68.06	-28.11
5	5	-50.41	-1.26

Seq 1	Seq 2	T_m (°C)	ΔG (kcal/mole)
Cycle 4			
0	0	-24.20	-2.53
0	1	63.83	-26.59
0	2	-39.09	-2.38
0	3	-18.48	-2.80
0	4	-4.99	-3.83
0	5	-72.09	-1.11
1	1	16.19	-7.34
1	2	-19.78	-2.56
1	3	-39.09	-2.38
1	4	-64.55	-1.43
1	5	-34.98	-2.91
2	2	18.68	-7.77
2	3	64.66	-26.88
2	4	-18.16	-3.75
2	5	-72.09	-1.11
3	3	18.68	-7.77
3	4	-21.26	-2.66
3	5	-35.89	-2.90
4	4	-59.28	-2.05
4	5	72.08	-29.34
5	5	-59.28	-2.05

3 Library Characterization with Gels

The primers on either end of the library sequences (Figure 1) interfere with tests for crosshybridizations by trapping duplexes into mismatched configurations. Thus, to remove the primers, an polymerase extension was done with a single RNA base at the end of the primer sequence. Then, RNAse was used to degrade the ribonucleotide, and thus, cut off a primer. Then, the protocol oligonucleotides could align in the desired hybridization frame (Figure 2). To track the progress of the protocol, a densitometer was use to compare intensities of different gel bands (Figure 2). The protocol products are in lanes 2 - 6, for 0 to 4 cycles of the protocol, respectively. The intensities of the bands in the dotted box, which contains the double-stranded molecules from the protocol (*i. e.* the extended product), were compared to the total intensity in the lane. Thus, the fraction of extended product is given by (Intensity of band in dotted box) divided by (Intensity of band in dotted box plus intensity of lower band). As the cycles progress, the proportion of selected product increases. Though the number of distinct sequences cannot be estimated from this technique, Figure 2 indicates that progressively larger fractions of the oligonucleotide population are being selected.

To further test the non-crosshybridizing properties of the manufactured sequences, a gel-based technique[9] to check for crosshybridization was applied

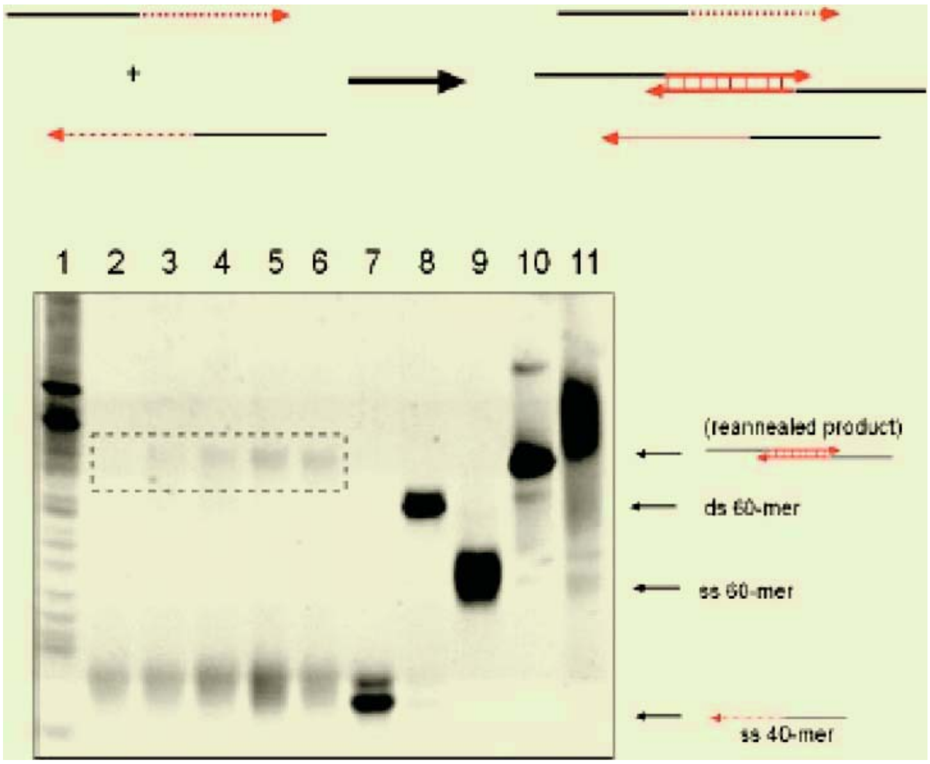
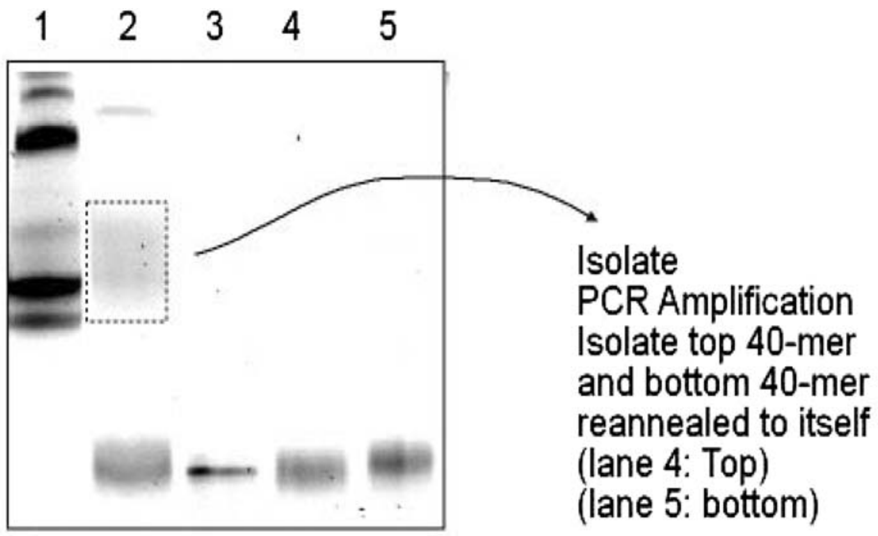


Fig. 2. Gel electrophoresis characterization of selection protocol progress. Protocol product for 0 to 4 cycles are in lanes 2 - 6, respectively. Using a densitometer, the percentage of reannealed product was 0% Cycle 0, 8.8% Cycle 1, 10.7% Cycle 2, 15.4% Cycle 3, and 22.9% Cycle 4. Lane 7 contains a single-stranded 40-mer marker, Lane 8 a double-stranded 60-mer marker, lane 9 a single-stranded 60-mer marker, and lane 10 the re-annealed protocol product from cycle 4

to the protocol product. The protocol product after 4 cycles was isolated, and PCR amplified. Then, the top strands and bottom strands were isolated from each other, and purified (Bottom gel in Figure 3). Then, separately, the top and bottom strands from the protocol product were allowed to anneal. These products are shown in lanes 4 and 5 of Figure 3. By comparing to the single-stranded marker in lane 3, the top and bottom strands exhibit no duplexes. In addition, the lack of a smear in the gel lanes indicates, that at least on the time scale of the electrophoresis, the top and bottom strands show no secondary structure.

4 Library Characterization with Spectroscopy

When oligonucleotides are heated, their ultraviolet absorbance at certain wavelengths increases. Double-stranded molecules undergo a phase transition as they



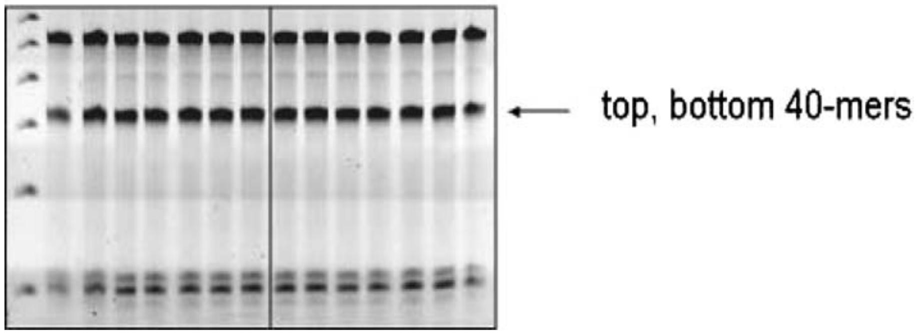


Fig. 3. Gel Electrophoresis results to test for crosshybridization. Lane 2 of top gel contains protocol product after 4 cycles of selection. The double-strands were extracted, and PCR amplified. Then, top and bottom strands were isolated and purified (Bottom Gel), and run separately in lanes 4 and 5. A single-stranded marker is in lane 3

become single-stranded, and a S shaped curve is observed in the UV absorbance at 260 nm as temperature increases. This melting curve gives an indication of the fraction of single-stranded base pairs in the mixture. In addition, as temperature increases, single-stranded oligonucleotides unstack, and a slow increase in absorbance is observed [14].

Melting curves were used to characterize the hybridization properties of the selection protocol product. The melting curves of protocol product (SP) were compared to curves from several standards, DNA samples of random 20-mers, forty non-crosshybridizing 20 mers (#1-#40) from [9], a Watson-

Crick complement pair of sequences (#5 and #41 from [9]), and a single-stranded oligonucleotide of length 20. The random 20-mers represent the starting population for the protocol, though without the primers. The non-crosshybridizing set from [9] and the single-stranded oligonucleotide give a standard for non-crosshybridization, and the Watson-Crick pair for hybridization.

DNA samples of selection products (SP) were prepared by PCR amplifying the selection products after appropriate cycles of the selection protocol, with DNA primers (Primer #1 (DP#1) = 5′ - CATCGAAGGGGT-GTTTTTT - 3′ and Primer #2 (DP#2) = 5′ - TCTTCATAAGTGATGC-CCG - 3′) to obtain enough concentration of DNA for T_m experiments. The amplified DNA samples were purified using the MiniElute PCR Purification Kit (QIAGEN Inc., Valencia, CA, USA) before T_m experiments. DNA was purchased from Integrated DNA Technologies (Coralville, IA). The purities of DNA samples were checked using denaturing TBE-urea polyacrylamide gel electrophoresis. T_m experiments were carried out using DU 800 UV/visible spectrophotometer equipped with the micro T_m analysis accessory (Beckman Coulter Inc., Futterton, CA, USA). The reaction mixtures for T_m experiments contain appropriate amounts of DNA samples, which give absorbance reading values at 260 nm of about 1.5, in Tris-EDTA buffer (pH 8.0;10 mM Tris and 1 mM Na2EDTA) with NaCl (1 M). Before T_m experiments, the reaction mixtures were degassed and equilibrated. Degassing was performed by quickly heating and cooling the reaction mixtures between 0 and 85°C. Equilibrations were performed by repeatedly heating from 25 to 95°C and holding at 95°C for 5 min until constant absorbance readings at 260 nm were achieved. Once equilibrated, T_m experiments were started at desired temperatures.

The results of the melting curve experiments are shown in Figure 4. The controls for the experiment were a single 20-mer (+), a Watson-Crick pair of 20-mers (□), and a set of 40 non-crosshybridizing 20-mers (△) that had been designed with the tool of [7] and experimentally tested in [9]. In addition, a set of random 20-mers (▼) were included, which approximated the starting population. The single 20-mer and the set of 40 non-crosshybridizing 20-mers are standards for non-crosshybridization. In addition, the set of random 20-mers, also, has little hybridization in it because of the complexity of the space and the slow kinetics to find Watson-Crick complements. These samples show a slow rate of increase indicating single-strand unstacking. The Watson-Crick pair represents a control for hybridization, exhibiting the expected *S* shaped curve. The top strands from Cycle 1 (▽) and 4 (■) of the protocol were extracted, and melting curves generated. These curves are similar to the single-stranded 20-mer, the 40 non-crosshybridizing 20-mers from [9], and the mixture of random 20-mers. This indicates that the top strands from cycles 1 and 4 are non-crosshybridizing, at least on the time scale of the melting curve. When

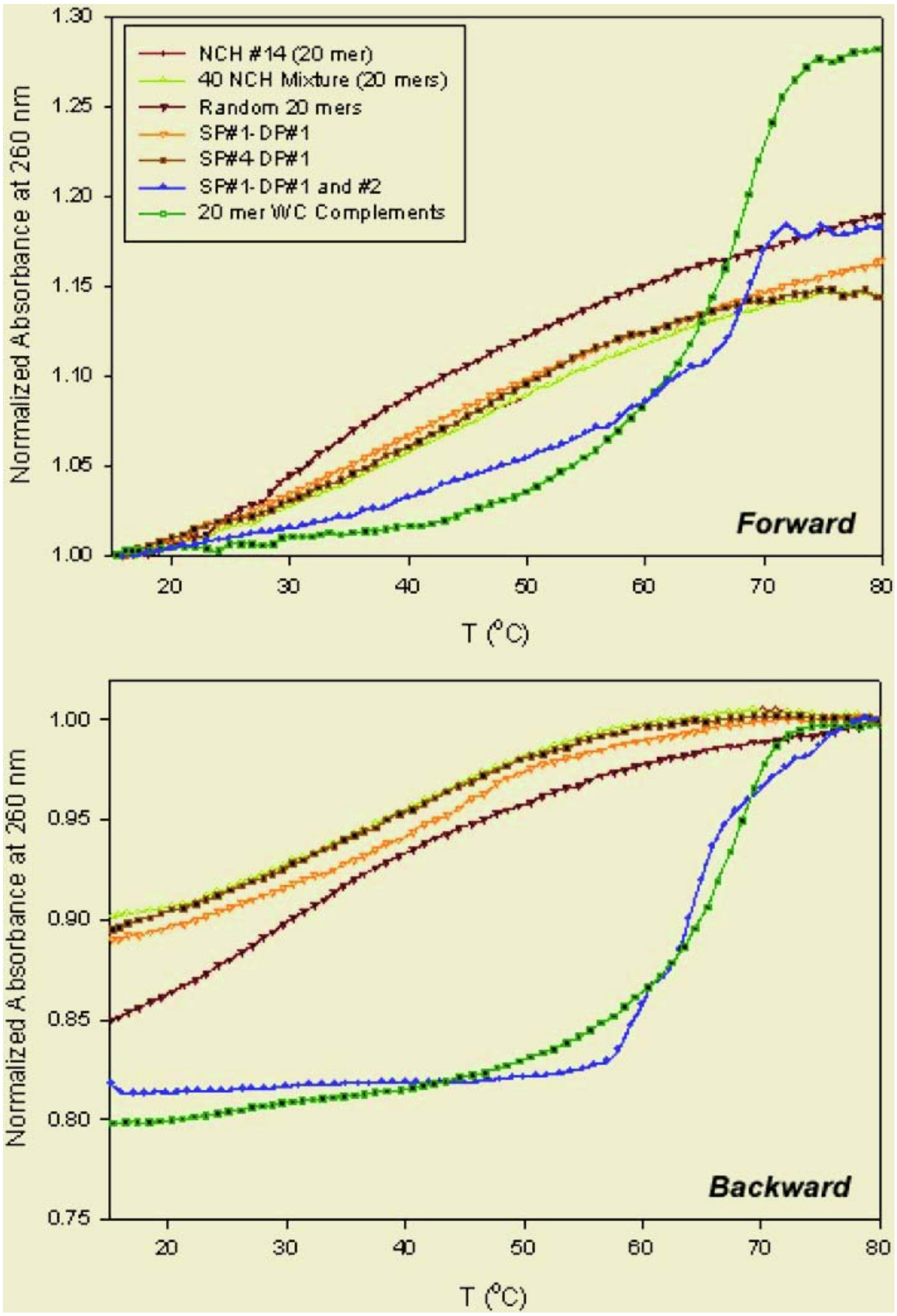


Fig. 4. Melting and annealing curves. Key is in Table 3

top and bottom strands from Cycle 1 (▲) were mixed together and measured, a curve similar to the Watson-Crick control was generated. This indicates that the primers are hybridizing, since the melting temperatures were close to the Watson-Crick 20-mer, and not a 60-mer, as expected if the whole molecule were duplex.

Table 3. Key for Figure 4

+	Non-crosshybridizing 20-mer
△	40 non-crosshybridizing 20-mers
▼	Mixture of random 20-mers
▽	Top Strands only from Cycle 1
■	Top Strands only from Cycle 4
▲	Top & Bottom Strands - Cycle 1
□	20-mer Watson-Crick Complements

In the forward direction (*i. e.* melting curve), the characteristic S-shaped curve for the melting transition was only observed for the Watson-Crick pair (□), and the combined top and bottom strands from cycle 1 (▲). The other curves showed a gentler increase in absorbance that might result from single-strand unstacking. In the backward direction (*i. e.* re-annealing), likewise the Watson-Crick pair and combined top and bottom strands from cycle 1 were the only curves exhibiting the S-shape. The top strands alone from cycles 1 and 4 (▽ and ■, respectively) were most similar to the single-stranded oligonucleotide (+) and the non-crosshybridizing library of 20-mers (△). Thus, the results indicate non-crosshybridization in the selected libraries.

5 Conclusion

Though any one test was not conclusive, the clone sequencing, gels, and melting curves all consistently indicated that the protocol was selecting for populations of non-crosshybridizing sequences. Even though in a starting population of random 20-mers, there is probably little annealing at all taking place, it does appear that the selection protocol selects a subset of the starting population, and progressively increases its size in subsequent iterations (Figure 2). This subset seems to be more non-crosshybridizing than the initial starting population and for increasing rounds of the protocol (Figure 4). Current efforts are focused on estimating the number of distinct sequences present in the population.

Acknowledgments

This work was supported by NSF Award EIA-0130385.

References

1. Chee, M., Yang, R., Hubbell, E., Berno, A., Huang, X.C., Stern, D., Winkler, J., Lockhart, D.J., Morris, M.S., Fodor, S.P.A.: Acessing genetic information with high-density DNA arrays. Science **274** (1996) 610–614
2. Adleman, L.M.: Molecular computation of solutions to combinatorial problems. Science **266** (1994) 1021–1024
3. Braich, R.S., Chelyapov, N., Johnson, C., Rothemund, P.W.K., Adleman, L.: Solution of a 20-variable 3-sat problem on a DNA computer. Science **296** (2002) 499–502
4. Mirkin, C., Letsinger, R.L., Mucic, R.C., Storhoff, J.J.: A DNA-based method for rationally assembling nanoparticles into macroscopic materials. Nature **382** (1996) 607–609
5. Winfree, E., Liu, F., Wenzler, L.A., Seeman, N.C.: Design and self-assembly of two-dimensional DNA crystals. Nature **394** (1998) 539–544
6. Mao, C., Sun, W., Shen, Z., Seeman, N.: A DNA nanomechanical device based on the b-z transition. Nature **397** (1997) 144–146
7. Deaton, R., Chen, J., Bi, H., Rose, J.A.: A software tool for generating non-crosshybridizing libraries of DNA oligonucleotides. [15] 252–261 Lecture Notes in Computer Science 2568.
8. Mauri, G., Ferretti, C.: Word design for molecular computing: a survey. In Chen, J., Reif, J., eds.: DNA Computing: 9th International Workshop on DNA-Based Computers, Berlin, University of Wisconsin-Madison, Madison, WI, June 2003, Springer-Verlag (2004) 37–46 Lecture Notes in Computer Science 2943.
9. Deaton, R., Kim, J.W., Chen, J.: Design and test of non-crosshybridizing oligonucleotide building blocks for DNA computers and nanostructures. Appl. Phys. Lett. **82** (2003) 1305–1307
10. Penchovsky, R., Ackermann, J.: DNA library design for molecular computation. Journal of Computational Biology **10** (2003) 215–229
11. Deaton, R., Chen, J., Bi, H., Garzon, M., Rubin, H., Wood, D.H.: A PCR-based protocol for *in vitro* selection of non-crosshybridizing oligonucleotides. [15] 196–204 Lecture Notes in Computer Science 2568.
12. Nuser, M., Deaton, R.: Simulations of dna computing with *in vitro* selection. Genetic Programming and Evolvable Machines **4** (2003) 173–183
13. SantaLucia, Jr., J.: A unified view of polymer, dumbbell, and oligonucleotide DNA nearest-neighbor thermodynamics. Proc. Natl. Acad. Sci. **95** (1998) 1460–1465
14. Bloomfield, V.A., Crothers, D.M., Tinoco Jr., I.: Nucleic Acids: Structures, Properties, and Functions. University Science Books, Sausalito, CA (2000)
15. Hagiya, M., Ohuchi, A., eds.: DNA Computing: 8th International Workshop on DNA-Based Computers, Berlin, University of Tokyo, Hokkaido University, Sapporo, Japan, June 2002, Springer-Verlag (2003) Lecture Notes in Computer Science 2568.

Error Free Self-assembly Using Error Prone Tiles

Ho-Lin Chen[1,⋆] and Ashish Goel[2,⋆⋆]

[1] Department of Computer Science, Stanford University
[2] Department of Management Science and Engineering and (by courtesy) Computer Science, Stanford University, Terman 311, Stanford CA 94305
{holin, ashishg}@stanford.edu

Abstract. DNA self-assembly is emerging as a key paradigm for nano-technology, nano-computation, and several related disciplines. In nature, DNA self-assembly is often equipped with explicit mechanisms for both error prevention and error correction. For artificial self-assembly, these problems are even more important since we are interested in assembling large systems with great precision.

We present an error-correction scheme, called snaked proof-reading, which can correct both growth and nucleation errors in a self-assembling system. This builds upon an earlier construction of Winfree and Bekbolatov [11], which could correct a limited class of growth errors. Like their construction, our system also replaces each tile in the system by a $k \times k$ block of tiles, and does not require changing the basic tile assembly model proposed by Rothemund and Winfree [8].

We perform a theoretical analysis of our system under fairly general assumptions: tiles can both attach and fall off depending on the thermodynamic rate parameters which also govern the error rate. We prove that with appropriate values of the block size, a seed row of n tiles can be extended into an $n \times n$ square of tiles *without errors* in expected time $\widetilde{O}(n)$, and further, this square remains stable for an expected time of $\widetilde{\Omega}(n)$. This is the first error-correction system for DNA self-assembly that has provably good assembly time (close to linear) and provable error-correction. The assembly time is the same, up to logarithmic factors, as the time for an irreversible, error-free assembly.

We also did a preliminary simulation study of our scheme. In simulations, our scheme performs much better (in terms of error-correction) than the earlier scheme of Winfree and Bekbolatov, and also much better than the unaltered tile system.

Our basic construction (and analysis) applies to all rectilinear tile systems (where growth happens from south to north and west to east). These systems include the Sierpinski tile system, the square-completion tile system, and the block cellular automata for simulating Turing machines. It also applies to counters, a basic primitive in many self-assembly constructions and computations.

⋆ Research supported in part by NSF Award 0323766.
⋆⋆ Research supported by NSF CAREER Award 0339262 and by NSF Award 0323766.

1 Introduction

Self-assembly is the ubiquitous process by which objects autonomously assemble into complexes. Nature provides many examples: Atoms react to form molecules. Molecules react to form crystals and supramolecules. Cells sometimes coalesce to form organisms. It is widely believed that self-assembly will ultimately become an important technology, enabling the fabrication of great quantities of small complex objects such as computer circuits. DNA has emerged as an important component to use in *artificial* self-assembly of nano-scale systems due to its small size, its incredible versatility, and the precedent set by the abundant use of DNA self-assembly in nature. Accordingly, DNA self-assembly has received significant attention over the last few years, both by practitioners [13, 15, 10, 11], and by theoreticians [5, 6, 12, 1, 7, 8, 2, 3, 4]. The theoretical results have focused on efficiently assembling structures of a controlled size (the canonical example being assembly of $n \times n$ squares) and shape. In this paper, we are interested in simultaneously achieving robustness and efficiency.

The Tile Assembly Model, originally proposed by Rothemund and Winfree [8], and later extended by Adleman *et al.* [2], provides a useful framework to study the *efficiency* (as opposed to robustness) of DNA self-assembly. In this model, a square tile is the basic unit of an assembly. Each tile has a glue on each side; each glue has a label and a strength (typically 1 or 2). A tile can attach to a position in an existing assembly if at all the edges where this tile “abuts” the assembly, the glues on the tile and the assembly are the same, and the total strength of these glues is at least equal to a system parameter called the temperature (typically 2). Assembly starts from a single seed crystal and proceeds by repeated accretion of single tiles. The speed of an addition (and hence the time for the entire process to complete) is determined by the concentrations of different tiles in the system. Details are in Section 2.

Rothemund and Winfree [8] gave an elegant self-assembling system for constructing squares by self-assembly in this model. Their construction of $n \times n$ squares requires time $\Theta(n \log n)$ and program size $\Theta(\log n)$. Adleman *et al.* [2] presented a new construction for assembling $n \times n$ squares which uses optimal time $\Theta(n)$ and optimal program size $\Theta(\frac{\log n}{\log \log n})$. Both constructions first assemble a roughly $\log n \times n$ rectangle (at temperature 2) by simulating a binary counter, and then complete the rectangle into a square. Later, Adleman *et al.* [3] studied several combinatorial optimization problems related to self-assembly. Together, the above results are a comprehensive treatment of the efficiency of self-assembly, but they do not address robustness.

In nature, DNA self-assembly is often equipped with explicit mechanisms for both error prevention and error correction. For artificial self-assembly, these problems are even more important since we are interested in assembling large systems with great precision. In reality, several effects are observed which lead to a loss of robustness compared to the model. The assembly tends to be reversible, i.e., tiles can fall away from an existing assembly. Also, incorrect tiles sometimes get incorporated and locked into a growing assembly, much like defects in a crystal. However, for sophisticated combinatorial assemblies like counters, which form the basis for controlling the size of a structure, a single error can lead to assemblies drastically larger or smaller (or different in other ways) than the intended structure. Finally, the temperature of the system can be controlled only imperfectly. Experimental studies of algorithmic self-assembly have observed error rates of 1% to 10% [11].

The work towards robustness has focused so far on two broad approaches. The first approach is to identify mechanisms used by nature for error-correction and error-prevention in DNA self-assembly and study how they can be leveraged in an algorithmic setting. One example of this approach is strand invasion [4]. The other approach is to design more combinatorial error-correction mechanisms. This is closest in spirit to the field of coding theory. One example of this approach, due to Winfree and Bekbolatov [11], is proof-reading tiles. They suggest replacing each tile in the original system with a $k \times k$ block. This provides some redundancy in the system (hence the loose analogy with coding theory). Their approach can correct *growth errors*, which result from an incorrect tile attaching at a correct location, i.e., a location where some other tile could have correctly attached. However, their approach does not reliably correct *nucleation errors*, which result from a tile (correct or incorrect) attaching at a site which is not yet active. Their proof-reading scheme is explained in section 3, along with the difference between growth and nucleation errors.

We present a modified proof-reading system which can correct both kinds of errors; we call it a snaked proof-reading system. Our scheme provably (under some mild assumptions) results in error-free assembly of an $n \times n$ square in time $\widetilde{O}(n)$ with high probability (whp). Further, our system results in the final assembly remaining stable for an $\Omega(n)$ duration whp. Hence, there is a large window during which there is a high probability of finding complete assemblies. The best-possible assembly time for an $n \times n$ structure is linear even without errors and even in the irreversible model. Thus, our system guarantees close to optimum speed. To the best of our knowledge, this is the first result which simultaneously achieves both robustness and efficiency.

Our snaked system is explained informally in section 3 using an illustrative example. We prove that the error-rate in this illustrative example is much better for our system than for that of Winfree and Bekbolatov. We give a formal description of our system in section 4 and prove the properties of error-correction and efficiency. Section 4 also provides simulation evidence with both our illustrative example and the Sierpinski tile system [10]; in both cases, we demonstrate that our system resulted in a significant reduction in errors.

Our analysis uses the thermodynamic model of Winfree [10]. We assume that the forward and reverse rates as well as the error-rates are governed by underlying thermodynamic parameters. We first analyze the performance of $k \times k$ proof-reading blocks in terms of the error-rate and efficiency, and then let k grow to $O(\log n)$. Our $\widetilde{O}$ notation hides polynomials in $\log n$. We believe that our analysis is slack, and can be significantly improved in terms of the dependence on k. We make some simplifying assumptions to allow our proofs to go through; our simulations indicate that these assumptions are just an artifact of our analysis and not really necessary.

Our basic construction (and analysis) applies to all rectilinear tile systems (where growth happens from south to north and west to east). These systems include the Sierpinski tile system, the square-completion tile system, and the block cellular automata for simulating Turing machines. It also applies to counters, a basic primitive in many self-assembly constructions and computations, but we omit the discussion about counters from this paper.

2 Tile Assembly Model

2.1 The Combinatorial Tile Assembly Model

The tile assembly model was originally proposed by Rothemund and Winfree[8, 2]. It extends the theoretical model of tiling by Wang [9] to include a mechanism for growth based on the physics of molecular self-assembly. Informally, each tile of an assembly is a square with glues of various types on each edge. Two tiles will stick to each other if they have compatible glues. We will present a succinct definition, with minor modifications for ease of explanation.

A tile is an oriented unit square with the north, east, south and west edges labeled from some alphabet Σ of glues. For each tile t, the labels of its four edges are denoted $\sigma_N(t)$, $\sigma_E(t)$, $\sigma_S(t)$, and $\sigma_W(t)$. Sometimes we will describe a tile t as the quadruple $(\sigma_N(t), \sigma_E(t), \sigma_S(t), \sigma_W(t))$. Consider the triple $< T, g, \tau >$ where T is a finite set of tiles, $\tau \in \mathbf{Z}_{>0}$ is the *temperature*, and g is the *glue strength* function from $\Sigma \times \Sigma$ to $\mathbf{Z}_{\geq 0}$, where Σ is the set of glues. It is assumed that for all $x, y \in \Sigma$, $(x \neq y)$ implies $g(x, y) = 0$ and there's a glue *null* $\in \Sigma$, such that $g(null, x) = 0$ for all $x \in \Sigma$. A *configuration* is a map from $\mathbf{Z}^2$ to $T \bigcup empty$.

Let C and D be two configurations. Suppose there exist some $t \in T$ and some $(x, y) \in \mathbf{Z}^2$ such that $D = C$ except at (x, y), $C(x, y) = null$ and $D(x, y) = t$. Let $f_{N,C,t}(x, y) = g(\sigma_N(t), \sigma_S(C(x, y+1))$. Informally $f_{N,C,t}(x, y)$ is the strength of the bond at the north side of t under configuration C. Define $f_{S,C,t}(x, y)$, $f_{E,C,t}(x, y)$ and $f_{W,C,t}(x, y)$ similarly. Then we say that tile t is *attachable* to C at position (x, y) iff $f_{N,C,t}(x, y) + f_{S,C,t}(x, y) + f_{E,C,t}(x, y) + f_{W,C,t}(x, y) \geq \tau$, and we write $C \rightarrow_{\mathbf{T}} D$ to denote the transition from C to D in attaching a tile to C at position (x, y). Informally, $C \rightarrow_{\mathbf{T}} D$ iff D can be obtained from C by adding a tile t such that the total strength of interaction between t and C is at least τ.

A *tile system* is a quadruple $\mathbf{T} =< T, s, g, \tau >$, where T, g, τ are as above and $s \in T$ is a special tile called the "seed". We define the notion of a *derived supertile* of a tile system $\mathbf{T} =< T, s, g, \tau >$ recursively as follows:

1. The configuration Γ such that $\Gamma(x, y) = \mathbf{empty}$ except when $(x, y) = (0, 0)$ and $\Gamma(0, 0) = s$ is a derived supertile of $\mathbf{T}$, and
2. if $C \rightarrow_{\mathbf{T}} D$ and C is a supertile of $\mathbf{T}$, then D is also a derived supertile of $\mathbf{T}$.

Informally, a derived supertile is either just the seed (condition 1 above), or obtained by legal addition of a single tile to another derived supertile (condition 2). We will often omit the word "derived" in the rest of the paper, and use the terms "seed supertile" or just "seed" or s to denote the special supertile in condition 1.

A *terminal supertile* of the tile system $\mathbf{T}$ is a derived supertile A such that there is no supertile B for which $A \rightarrow_{\mathbf{T}} B$. If there is a terminal supertile A such that for any derived supertile B, $B \rightarrow^*_{\mathbf{T}} A$, we say that the tile system *uniquely produces* A. Given a tile system $\mathbf{T}$ which uniquely produces a supertile, we say that the program size complexity of the system is $|T|$ i.e. the number of tile types.

2.2 The Kinetic Model: Rates and Free Energy

Adleman *et al.* presented a model for running time of reversible self-assemblies [2]. In this paper, we use a kinetic model proposed by Winfree which computes the forward and reversed rate as functions of thermodynamic parameters [10]. It has the following assumptions:

1. Tile concentrations are held constant throughout the self-assembly process.
2. Supertiles do not interact with each other. The only two reactions allowed are addition of a tile to a supertile, and the dissociation of a tile from a supertile.
3. The forward rate constants for all tiles are identical.
4. The reverse rate depends exponentially on the number of base-pair bonds which must be broken, and the mismatched sticky ends make no base-pair bonds.

There are two free parameters in this model, both of which are dimensionless free energies: $G_{mc} > 0$ measures the entropic cost of putting a tile at a binding site and depends on the tile concentration, $G_{se} > 0$ measures the free energy cost of breaking a single strength-1 bond. Under this model, we can approximate the forward and reverse rates for each of the tile-supertile reactions in the process of self-assembly of DNA tiles as follows:

The rate of addition of a tile to a supertile, f, is $pe^{-G_{mc}}$.

The rate of dissociation of a tile from a supertile, r_b, is $pe^{-bG_{se}}$, where b is the strength with which the tile is attached to the supertiles.

The parameter p simply gives us the time scale for the self-assembly.

Winfree suggests using G_{mc} just a little smaller than $2G_{se}$ for self-assembly at temperature two. We use the same operating region.

3 An Illustrative Example

While the ideas that we develop in this section are applicable to general self-assemblies, a simple one dimensional example will be used for illustrative purposes. The tile system is one that can compute the parity of a bit string and we will refer to it as the parity system. The tiles are essentially a simplification of the tiles in the Sierpinski tile system [10] and are obtained by making the top side of each tile in the Sierpinski system inert. The tiles for the parity system are illustrated below in figure 1(a). The temperature is 2. The "input" will consist of a structure of $n + 2$ tiles. The "input" tiles are assumed to be arranged in two rows. The bottom row has $n + 1$ tiles. The rightmost tile on the bottom row is inert on the right, the leftmost is inert on the left, and they are all inert on the bottom. Each tile in the bottom row except the leftmost has a glue labeled either 0 or 1 on the top. The second row has just one input tile, sitting on top of the leftmost tile in the bottom row. This second row tile is inert on the left and the top, and has a glue labeled 0 on the right. Thus the input codes a string of n bits. With this input, the tiles in the parity system will form a layer covering the n exposed glues in the bottom row. Further, the rightmost tile in the top row will leave a glue labeled 0 on the right if the parity of the bit string is 0 (i.e. the number of ones is even) and 1 otherwise. Figure 1(b) illustrates this construction for $n = 4$ and the input string 1111. The glues are written on the edges of the tiles and the input tiles are shaded.

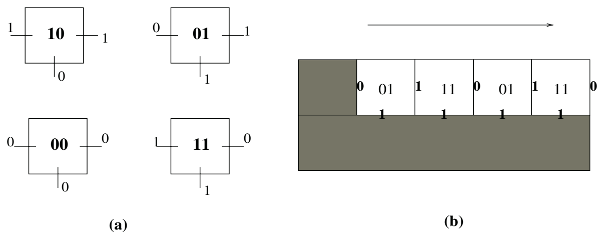


Fig. 1. (a) The parity tile system. (b) Illustrating the action of the parity tile system on the ”input” string 1111. The arrow at the top represents the order in which tiles must attach in the absence of errors

In this setting, tiles in the top row attach from left to right, if there are no errors. Hence, in the absence of errors, there is always a correct “next” position

3.1 Growth Errors and the Winfree-Bekbolatov Proof-Reading System

An error is said to be a “growth” [11] error if an incorrect tile attaches in the next position. The proof-reading approach of Winfree and Bekbolatov [11] can correct such errors by using redundancy. They replace each tile in the system with four tiles, arranged in a 2×2 block. Figure 2(b) depicts the four tiles that replace a 10 tile. The glues internal to the block are all unique. This added redundancy results in resilience to growth errors. The details are described in their paper.

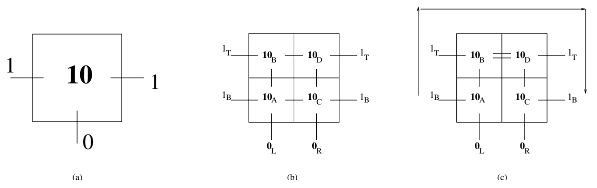


Fig. 2. (a) The original 10 tile. (b) The four proof-reading tiles for the 10 tile, using the construction of Winfree and Bekbolatov [11]. (c) The snaked proof-reading tiles for the parity tile system. The internal glues are all unique to the 2×2 block corresponding to the 10 tile. Notice that there is no glue on the right side of 10_A or the left side of 10_C and that the glue between the top two tiles is of strength 2. This means that the assembly process doubles or “snakes” back onto itself, as demonstrated by the arrow

3.2 Nucleation Errors and Improved Proof-Reading

However, there is another, more insidious kind of error that can happen. A tile may attach at a position other than the correct “next” position using just a strength one glue. This would be the incorrect tile, and hence an error with probability 50%, and such an error will propagate to the right ad infinitum even if we are using the proof-reading tile set of Winfree and Bekbolatov. We call such errors “nucleation” errors[1]. In more

[1] Winfree and Bekbolatov call these facet roughening errors and reserve the term nucleation errors for another phenomenon.

complicated systems, these errors can also happen on the boundary of a completed assembly, making it very hard to precisely control the size of an assembly. Both growth errors and nucleation errors are caused by what we term an insufficient attachment – the attachment of a tile to an existing assembly using a total glue strength of only 1 (even though the temperature is 2) and then being "stabilized" (i.e. held by strength 2) by another tile attaching in the vicinity. Insufficient attachments are unlikely at any given site (say they happen with probability x) but over the course of n attachments, the probability of getting at least one insufficient attachment may become as large as $O(nx)$. We will now show a design that requires two insufficient attachments in close proximity to have an error that can propagate, and significantly reduces the chances of getting an error (either growth or nucleation). Figure 2(c) shows the 2 $\times$ 2 block that replaces a single tile (say tile 10), and the arrow shows the order in which the sub-tiles attach at a site when there have been no insufficient attachments. Notice that there is no glue between tiles 10_A and 10_C. This is what prevents nucleation errors from propagating without another insufficient attachment. We call this the "snaked" proof-reading system, since the assembly process for a block doubles back on itself.

It is easy to show that the above approach can be extended to arbitrary $k \times k$ sized blocks, to get lower and lower error rates. The above idea can also be extended to Sierpinski tile systems [10] and counters [8, 2], though for technical reasons, a 3 $\times$ 3 block is needed at a minimum to take care of nucleation errors in these more complicated systems. Detailed analysis is given in section 4. However, the following lemmas are useful to illustrate the kind of improvements we can expect to get. The quantities f, r and G_{se} are as defined in section 2.2. An *insufficient attachment* at temperature two is the process that a tile attaches with strength one, but, before it falls off, another tile attaches right next to it and both tiles are held by strength at least two.

Lemma 1. *The rate at which an insufficient attachment happens at any location in a growing assembly is* $\frac{f^2}{r}e^{-G_{se}} = O(e^{-3G_{se}})$.

Proof. The rate of an insufficient attachment can be modeled as the Markov Chain shown in figure 3. For a nucleation error to happen, first a single tile must attach (at rate f). The fall-off rate of the first tile is $re^{G_{se}}$ and the rate at which a second tile can come and attach to the first tile is f. After the second tile attaches, an insufficient attachment has happened. So the overall rate of an insufficient attachment is $f * \frac{f}{f+re^{G_{se}}} \approx \frac{f^2}{r}e^{-G_{se}}$

Without proof-reading, or even using the proof-reading system of Winfree and Bekbolatov, a single insufficient attachment can cause a nucleation error, and hence the

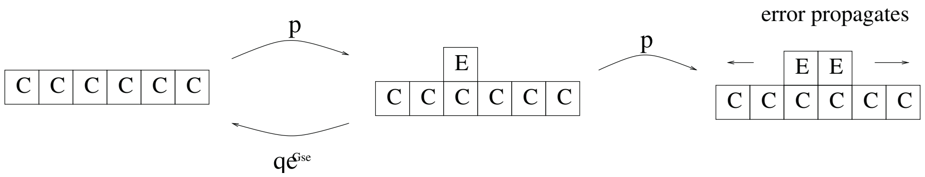


Fig. 3. The C tiles represent the existing assembly, and the E tiles are new erroneous tiles

rate of nucleation error at any location is also $O(e^{-3G_{se}})$. The next lemma shows the improvement obtained using our snaked proof-reading system. The difference is even more pronounced if we compare the nucleation error rate to the growth rate, which is a natural measurement unit in this system. The ratio of the nucleation error rate to the growth rate is $O(e^{-G_{se}})$ in the original proof-reading system, whereas it is $O(e^{-2G_{se}})$ in our system, a quadratic improvement.

Lemma 2. *The rate at which a nucleation error takes place in our snaked proof-reading system is $O(e^{-4G_{se}})$.*

Proof. In the snaked system, two insufficient attachments need to happen next to each other for a nucleation error to occur. According to lemma 1, the first insufficient attachment happens at rate $O(e^{-3G_{se}})$. After the first insufficient attachment, the error will eventually be corrected unless another insufficient attachment happens next to the first. The second insufficient attachment happens at rate $O(e^{-3G_{se}})$; but the earlier insufficient attachment gets "corrected" at rate $O(e^{-2G_{se}})$ (remember that $a \approx 1$ and hence a tile attached with strength 2 falls off at roughly the growth rate). Hence, the probability of another insufficient attachment taking place before the previous insufficient attachment gets reversed is $O(e^{-G_{se}})$, bringing the nucleation error rate down to $O(e^{-4G_{se}})$.

For growth errors, the proof-reading system of Winfree and Bekbolatov achieves a reduced error rate of $O(e^{-4G_{se}})$, a property preserved by our modification.

4 The General Snaked Proofreading System

The system shown in the previous section only works for prevention of nucleation errors in one direction (west to east). The system we describe in this section can improve any rectilinear tile system [2] and prevents nucleation errors in both growth directions.

First, we look at rectilinear systems in which all glues have strength 1. To improve this kind of system, each tile T in the original system is replaced by a $2k \times 2k$ block ($k \geq 2$) $T_{1,1}$, $T_{1,2}$, ..., $T_{2k,2k}$. Each glue G_i in the original system is replaced by $2k$ glues $G_{i,1}, G_{i,2}, \ldots, G_{i,2k}$ with strength 1 on the corresponding boundary of the block. All glues internal to the block have strength 1 except the following:

1. The east sides of tiles $T_{1,2i-1}$ are inert, as well as the west sides of tiles $T_{1,2i}$ for $i = 1, 2, \ldots, k-1$.
2. The north sides of tiles $T_{2i,1}$ are inert, as well as the south sides of tiles $T_{1,2i+1}$ for $i = 1, 2, \ldots, k-1$.
3. The glues on the north sides of tiles $T_{2i,2i+1}$ have strength 2, as well as the glues on the south sides of tiles $T_{2i+1,2i+1}$ for $i = 1, 2, \ldots, k-1$.
4. The glues on the east sides of tiles $T_{2i,2i-1}$ have strength 2, as well as the glues on the west sides of tiles $T_{2i,2i}$ for $i = 1, 2, \ldots, k$.

[2] A rectilinear tile system is one where growth occurs in a rectilinear fashion - from south to north and from west to east.

5. The east side of the tile $T_{2k-2,2k-1}$ is inert, as well as the west side of the tile $T_{2k-2,2k}$.
6. The glue on the north side of the tile $T_{2k-2,2k}$ has strength 2, as well as the south side of the tile $T_{2k-1,2k}$.

The glues internal to the block are unique to that block and don't appear on any other blocks. Informally, the blocks attach to each other using the same logic as the original system.

An illustrative example with $k = 2$ is shown in figure 4(a). The numbering of the tiles in figure 4(b) denotes the sequence of the tile attachment in the assembly process. It is worth noticing that all the tiles on the northern and eastern side of the block are held by strength at least 3. So whenever all the tiles on a block are attached, it is unlikely for them to fall off.

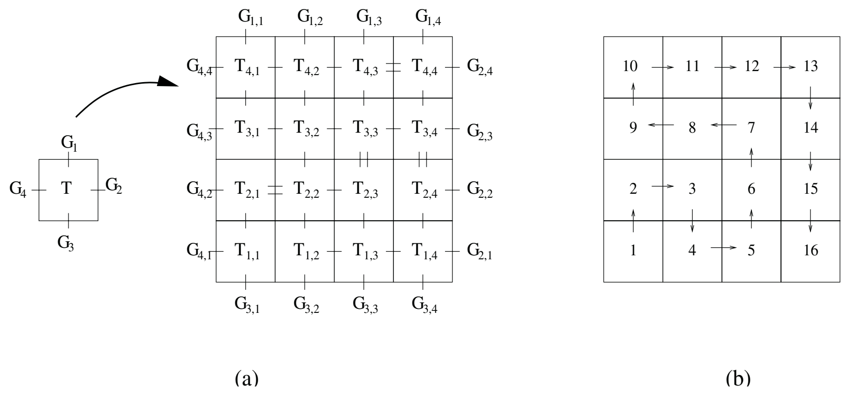


Fig. 4. (a) The structure of 4x4 block. (b) The order of the growth

Recall that f denoted the forward rate of a tile attaching, and r denotes the backward rate of a tile held by strength 2 falling off. In the rest of the section, we assume that $f = r$ and tiles held by strength three do not fall off. We need to make this assumption for our proof to go through, but we don't believe they are necessary.

Here are some definitions we will use in this section: a *k-bottleneck* is a connected structure which requires at least k insufficient attachments to form. A *block error* occurs if all the tiles in a block have attached and are all incorrect (compared to perfect growth).It is easy to prove that a block error is just an example of a k-bottleneck.

We are going to consider an idealized system where the south and west boundary is already assembled and the tiles in the square are going to assemble in a rectilinear fashion. The following theorems represent our main analytical result:

Theorem 1. *With a $2k \times 2k$ snaked tile system (for some fixed k), assuming we can set $e^{G_{se}}$ to be $O(n^{\frac{2}{k}})$, an $n \times n$ square of blocks can be assembled in time $O(n^{1+\frac{4}{k}})$ and with high probability, no block errors happen $\Omega(n^{1+\frac{4}{k}})$ time after that.*

Theorem 2. *With a $2k \times 2k$ snaked tile system, $k = O(\log n)$, assuming that we can set $e^{G_{se}}$ to be $O(k^6)$, an $n \times n$ square of blocks can be assembled in time $\widetilde{O}(n)$ and with high probability, no block errors happen for $\widetilde{\Omega}(n)$ time after that.*

Here, the $\widetilde{O}$ notation hides factors which are polynomial in k and $\log n$. Informally, Theorems 1 and 2 say that snaked proofreading results in tile systems which assembly quickly and remain stable for a long time.

In fact, we believe that our scheme achieves good performance without having to set $e^{G_{se}}$ to be as high as $O(k^6)$ and the ratio between forward and backward rate can be set to some constant for getting a good performance. The simulation results are described at the end of this section confirm our intuition.

4.1 Proof of the Main Result

Since all tiles in a correctly attached block are held by strength three, once a correct block attaches at a location, none of its tiles ever fall off. So, it suffices to only consider the perimeter of the supertile. For ease of exposition, we are going to focus on errors that happen on the east edge of the assembly.

Lemma 3. *Consider any connected structure caused by m insufficient attachments ($1 \leq m \leq k$). Then the width of the structure can be at most $2m$, and the height of the structure can be at most $2k$ (i.e., this connected structure can only span two blocks). This structure will fall off in expected time $O(\frac{k^5}{r})$ unless there's a block error somewhere in the assembly or an insufficient attachment happens within the (at most two) blocks spanned by the structure.*

PROOF OUTLINE: The proof of this lemma involves a lot of technical details. Due to space constraints, we only present a sketch in this version. In the structure of $2k \times 2k$ snaked tiles, all the glues between the $(2i)$-th row and $(2i+1)$-th row have strength 1 for all i. So, to increase the width from $2i$ to $2i + 1$, we must have at least one insufficient attachment. So, with m insufficient attachments, the width of the structure can be at most $2m$. Using similar arguments, the height of the structure can be at most $2k$. Also, the attached tiles can be partitioned into $O(k)$ parts. Each of these parts can be viewed as a $2 \times O(k)$ rectangle with every internal glue having strength 1. The process of tiles attaching to and detaching from each rectangle can be modeled using two orthogonal random walks and hence, each rectangle will fall off in expected time $O(\frac{k^4}{r})$. The different rectangles can fall off sequentially, and after one rectangle falls off completely, none of its tiles will attach again unless an insufficient attachment happens. Thus, the structure will fall off in expected time $O(\frac{k^5}{r})$ unless there's a block error (anywhere in the assembly) or an insufficient attachment happens (within the two blocks) before the structure has a chance to fall off. □

Theorem 3. *Assume that we use a $2k \times 2k$ snaked tile system and $G_{mc} = 2G_{se}$. Then for any ϵ, there exists a constant c such that, with probability $1 - \epsilon$, no k-bottleneck will happen at a specific location within time $c\frac{1}{f}e^{G_{se}}(\frac{e^{-G_{se}}+1/k^6}{e^{-G_{se}}})^{k-1}$.*

Proof. By definition, k insufficient attachments are required before a k-bottleneck happens. After i insufficient attachments take place, one of the following is going to happen:

– One more insufficient attachment. Consider any structure X caused by i insufficient attachments. By Lemma 3, the size of X cannot exceed two blocks, hence the number of insufficient attachment locations that can cause this structure to grow larger is at most $4k$. So, the rate of the $(i+1)$-th insufficient attachment happening is at most $4kfe^{-G_{se}}$.
– All the attached tiles fall off. By Lemma 3, the expected time for all the attached tiles to fall off is $O(\frac{k^5}{r})$

So, after i insufficient attachments happen, the probability of the $(i+1)$-th insufficient attachment happening before all tiles fall off is $O(\frac{kfe^{-G_{se}}}{kfe^{-G_{se}}+r/k^5}) = O(\frac{e^{-G_{se}}}{e^{-G_{se}}+1/k^6})$. So, after the first insufficient attachment takes place, the probability of a k-bottleneck happening before all the attached tiles fall off is less than $O((\frac{e^{-G_{se}}}{e^{-G_{se}}+1/k^6})^{k-1})$. As shown in Lemma 1, the expected time for the first insufficient attachment is $O(\frac{1}{f}e^{G_{se}})$. So, the expected time for a k-bottleneck to happen at a certain location is at most $O(\frac{1}{f}e^{G_{se}}(\frac{e^{-G_{se}}+1/k^6}{e^{-G_{se}}})^{k-1})$. Hence, for any small ϵ, we can find a constant c such that, with probability $1-\epsilon$, no k-bottleneck will happen at a specific location within time $c\frac{1}{f}e^{G_{se}}(\frac{e^{-G_{se}}+1/k^6}{e^{-G_{se}}})^{k-1}$.

Theorem 4. *If we assume there are no k-bottlenecks, and the rate of insufficient attachments is at most $O(\frac{f}{k^6})$, then an $n \times n$ square of $2k \times 2k$ snaked tile blocks can be assembled in expected time $O(\frac{k^5 n}{f})$.*

Proof. With the snaked tile system, after all the tiles in a block attach, all the tiles are held by strength at least 3 and will never fall off. Using the running time analysis technique of Adleman *et al.* [2], the system finishes in expected time $O(n \times T_B)$, where n is the size of the terminal shape and T_B is the expected time for a block to assemble. Without presence of k-bottlenecks, when we want to assemble a block, the erroneous tiles that currently occupy that block are formed by at most $k-1$ insufficient attachments. By Lemma 3, without any further insufficient attachments happening, the erroneous tiles will fall off in time $O(\frac{k^5}{f})$ and the correct block can attach within time $O(\frac{k^4}{f})$. By assumption, the rate of insufficient attachment happening is at most $O(\frac{f}{k^6})$, and there are at most $O(k)$ locations for insufficient attachments to happen and affect this process. So, there's a constant probability that no insufficient attachments will happen during the whole process and thus the time required to assemble a block, T_B, is at most $O(\frac{k^5}{f})$.

Theorems 1 and 2 follow from the above two theorems. Notice that there is a lot of slack in our analysis.

4.2 Simulation Results

We use the simulation program xgrow written by Winfree *et al.* [14].

We first use three different systems to build a square of 20×20 blocks with the Sierpinski pattern [10]. The column "snaked" refers to the system described in this paper; the column "proofreading" corresponds to the original proofreading system described by Winfree and Bekbolatov; the column "original" refers to the system without any error correction. The results are summarized in table 1. The block size for our snaked system as well as the original proofreading system is 4×4. Similar results were observed for a wide range of simulation scenarios. As is clear, our snaked tile system has a much lower error rate, has a much higher stability time, and is only two-three times slower than the proofreading system of Winfree and Bekbolatov [11]. The original system only needs to assemble a much smaller structure (since it uses a 1×1 block); hence the "reversal" in the error-rates for the original and the proofreading system in the second simulation.

Table 1. Assembling a 20×20 Sierpinski block. A stability time of 0 indicates that the final square became unstable (i.e., an extra block of tiles attached on the periphery of the desired supertile) even before the complete supertile formed. The values represent averages over 100 runs

	$G_{mc} = 15, G_{se} = 7.8$			$G_{mc} = 15, G_{se} = 8.0$		
	Original	Proofreading	Snaked	Original	Proofreading	Snaked
Time to assemble (seconds)	550	2230	6020	350	1750	3780
Error Probability	52%	24%	0%	63%	75%	0%
Time it remains stable (seconds) after completion	0	0	>400000	0	0	5700

Also, for the 2×2 snaked tile system and the original proofreading system, we took a straight line boundary of 200 tiles (i.e., 100 blocks) and tested the average time (in seconds; virtual time) for a block error to happen under different G_{se}'s. Here, $2G_{se} - G_{mc}$ is set to be 0.2. Theoretically, the expected time for a block error to happen in the snaked tile system is $O(e^{4G_{se}})$, and the expected time for a block error to happen in the proofreading system is $O(e^{3G_{se}})$. The result is shown in figure 5(a); the y-axis uses a log-scale. Clearly, the slope of the curve for the snaked tile system confirms our analysis – the slope is very close to 4, and significantly more than the slope for the original proofreading system. For the larger values of G_{se}, we could only plot the results for the original proof-reading system, since the simulator did not report any errors with the snaked tile system for the time scales over which we conducted the simulation.

We also tested the error rate for parity systems of different seed lengths. We called an experiment an error if the final supertile was different from the one we expect in the absence of errors. We used $G_{se} = 7.0, G_{mc} = 13.6$. The result is shown in figure 5(b); again, a significant reduction in error rate is observed. For both figures 5(a) and 5(b), qualitatively similar results were observed for widely varying simulation parameters.

Our simulation results show that our analysis is very close to reality even without idealized parameter conditions. For example, we did not use $G_{mc} = 2G_{se}$ but instead used G_{mc} slightly smaller than $2G_{se}$ as suggested by Winfree [10]. Also, the simulator

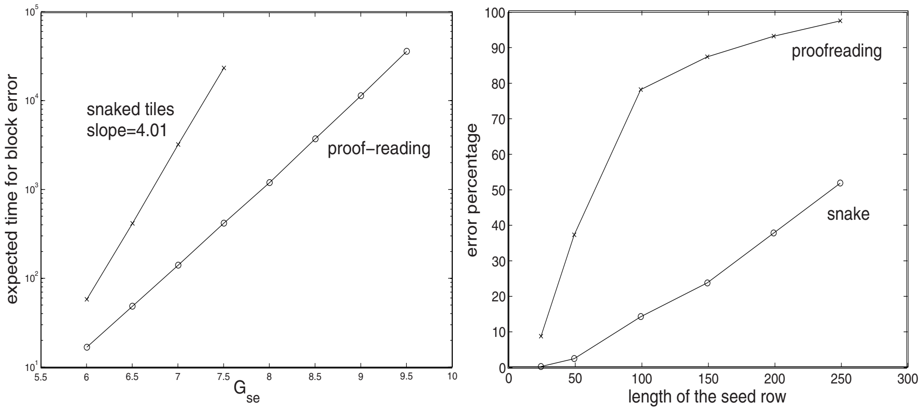


Fig. 5. (a) The first figure shows the relation between the average time for a block error to happen and G_{se} (average over 100 runs). (b) The second figure shows that the error rate for snaked tiles is much smaller than proofreading tiles (average over 200 runs)

allows tiles held by strength 3 to fall off, contrary to our assumption. Thus, we believe that our snaked system works much better (and under a much wider set of conditions) than we have been able to formally prove.

5 Future Directions

It would be interesting to extend our analysis to remove some of our assumptions. Also, we believe that the total assembly time for our system should just be $O(k^2 n)$ for assembling an $n \times n$ square using $k \times k$ snaked blocks. One of the biggest bottlenecks in proving this bound is an analysis of the assembly time of an $n \times n$ square assuming that there are no errors but that the system is reversible, i.e., tiles can both attach and detach. We believe that the assembly time for this system should be $O(n)$ along the lines of the irreversible system [2], but have not yet been able to prove it.

Acknowledgments

We would like to thank Qi Cheng and Erik Winfree for many useful discussions. We would also like to thank Erik Winfree for loaning us his tile simulator, xgrow, which was an indispensable aid in this research.

References

1. L. Adleman. Towards a mathematical theory of self-assembly. Technical Report 00-722, Department of Computer Science, University of Southern California, 2000.
2. L. Adleman, Q. Cheng, A. Goel, and M.-D. Huang. Running time and program size for self-assembled squares. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 740–748. ACM Press, 2001.

3. L. Adleman, Q. Cheng, A. Goel, M.-D. Huang, D. Kempe, P. Moisset de Espans, and P. Rothemund. Combinatorial optimization problems in self-assembly. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 23–32. ACM Press, 2002.
4. H. Chen, Q. Cheng, A. Goel, M.-D. Huang, and P. Moisset de Espans. Invadable self-assembly: Combining robustness with efficiency. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 883–892, 2004.
5. M. Lagoudakis and T. LaBean. 2D DNA self-assembly for satisfiability. In *Proceedings of the 5th DIMACS Workshop on DNA Based Computers in DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 54. MIT: Cambridge, 1999.
6. J. Reif. Local parallel biomolecular computation. In H. Rubin, editor, *Third Annual DIMACS Workshop on DNA Based Computers, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 1998.
7. P. Rothemund. *Theory and Experiments in Algorithmic Self-Assembly*. PhD thesis, University of Southern California, 2001.
8. P. Rothemund and E. Winfree. The program-size complexity of self-assembled squares (extended abstract). In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 459–468. ACM Press, 2000.
9. H. Wang. Proving theorems by pattern recognition II. Bell Systems Technical Journal, 1961. 40:1-42.
10. E. Winfree. *Algorithmic Self-Assembly of DNA*. PhD thesis, California Institute of Technology, Pasadena, 1998.
11. E. Winfree and R. Bekbolatov. Proofreading tile sets: Error correction for algorithmic self-assembly. In *Proceedings of the Ninth International Meeting on DNA Based Computers*. Madison, Wisconsin, June 2003.
12. E. Winfree, F. Liu, L. Wenzler, and N. Seeman. Design and self-assembly of two-dimensional DNA crystals. *Nature*, (394):539–544, Aug 1998.
13. E. Winfree, X. Yang, and N. Seeman. Universal computation via self-assembly of DNA: Some theory and experiments. In *Proceedings of the Second Annual Meeting on DNA Based Computers*. Princeton University, June 1996.
14. E. Winfree *et al.*. The xgrow simulator. *http://www.dna.caltech.edu/Xgrow/xgrow_www.html*.
15. B. Yurke, A. Turberfield, A. Mills Jr, F. Simmel, and J. Neumann. A DNA-fuelled molecular machine made of DNA. *Nature*, (406):605–608, Aug 2000.

On the Computational Complexity of P Automata*

Erzsébet Csuhaj-Varjú[1], Oscar H. Ibarra[2], and György Vaszil[1]

[1] Computer and Automation Research Institute, Hungarian Academy of Sciences,
Kende utca 13-17, 1111 Budapest, Hungary
{csuhaj, vaszil}@sztaki.hu

[2] Department of Computer Science, University of California,
Santa Barbara, CA 93106, USA
ibarra@cs.ucsb.edu

Abstract. We characterize the classes of languages described by P automata, i.e., accepting P systems with communication rules only. Motivated by properties of natural computing systems, we study computational complexity classes with a certain restriction on the use of the available workspace in the course of computations and relate these to the language classes described by P automata. We prove that if the rules of the P system are applied sequentially, then the accepted language class is strictly included in the class of languages accepted by one-way Turing machines with a logarithmically bounded workspace, and if the rules are applied in the maximal parallel manner, then the class of context-sensitive languages is obtained.

1 Introduction

Membrane systems, or P systems, are biomolecular computing devices working in a distributed and parallel manner inspired by the functioning of the living cell. The main ingredient of a P system is a hierarchically embedded structure of membranes with rules associated to the regions describing the evolution of the objects present in the membranes. The evolution of the system corresponds to a computation. P systems have intensively been studied in the past few years, the interested reader might consult the monograph [10] for a systematic study of the area.

The introduction of P automata in [1] was motivated by an idea recently attracting researchers, namely, to use P systems as language acceptors. The objects in a P automaton may move through the membranes from region to region, but they may not be modified during the functioning of the systems, and furthermore, the described languages are obtained as the set of accepted sequences of

* The research of E. Csuhaj-Varjú and Gy. Vaszil was supported in part by the Hungarian Scientific Research Fund "OTKA" grant no. T 042529. The research of O. H. Ibarra was supported in part by NSF Grants IIS-0101134 and CCR02-08595.

multisets containing the objects entering from the environment during the evolution of the system. The environment is considered to have an infinite supply of objects, any number of symbols may be requested by the application of one or more rules associated to the skin membrane, and these symbols may traverse this membrane and enter when they are requested to do so.

A result on the accepting power of P automata was already established in [1] stating that for any recursively enumerable language, there is a P automaton accepting the image of the language under a certain mapping. Similar results were also obtained in [3], [4], and [7], for P automata with different features and different mappings to obtain the recursively enumerable language.

In the present paper we continue the study of the power of P automata, but this time we are interested in the exact characterization of the languages of the multiset sequences entering the system through the skin membrane during a computation. We do this by establishing a correspondence between the symbols of an alphabet and the multisets that might ever enter the P automaton, and characterize the set of words corresponding to the set of accepted multiset sequences. This approach differs from the ones mentioned above because we do not allow erasing, that is, each nonempty multiset corresponds to a symbol of the alphabet when defining the string represented by the multiset sequence. This means that the workspace of the P automaton is provided only by the objects of the input obtained from the environment during a computation, which is a very natural way of restricting the use of the resources: As the processing of the input progresses, additional parts of the workspace become available with each step. Thus, the workspace which can be used in the course of the computation is provided in accordance with the number of symbols actually read from the input, in other words, the computation is made possible by manipulating what is already obtained from the input. This idea agrees very well with the behaviour of natural systems where the result of a computation is also obtained by using the resources provided by the input, the object of the computation itself.

We consider the sequential and the so-called maximal parallel way of rule application. In the sequential case, the number of different multisets that may ever enter the system is finite which means that there is a natural one-to-one correspondence between these multisets and the symbols of a finite alphabet. This is not necessarily so when the rules are applied in the maximal parallel way, in this case P automata can be considered as devices accepting finite strings over an infinite alphabet. In this paper we do not study the case of infinite alphabets, we use instead a mapping that maps the infinite set of different multisets to a finite alphabet, thus we will be able to speak of languages accepted by P automata using the rules in the sequential or in the maximal parallel manner, the languages being in both cases over a finite alphabet.

We show that the languages which can be characterized by P automata in this sense using the rules in the sequential manner are strictly included in the class of languages accepted by one-way Turing machines using logarithmically bounded workspace, while if the rules are used in the maximal parallel way, the class of context-sensitive languages is obtained.

2 Definitions

We first recall the notions and the notations we use. Let V be an alphabet, let V^* be the set of all words over V, and let $V^+ = V^* - \{\varepsilon\}$ where ε denotes the empty word. We denote the length of a word $w \in V^*$ by $|w|$, and the number of occurrences of a symbol $a \in V$ in w by $|w|_a$. The set of natural numbers is denoted by $\mathbb{N}$.

A multiset is a pair $M = (V, f)$, where V is an arbitrary (not necessarily finite) set of objects and $f : V \to \mathbb{N}$ is a mapping which assigns to each object its multiplicity. The support of $M = (V, f)$ is the set $supp(M) = \{a \in V \mid f(a) \geq 1\}$. If V is a finite set, then M is called a finite multiset. The set of all finite multisets over the set V is denoted by V°.

We say that $a \in M = (V, f)$ if $a \in supp(M)$, and $M_1 = (V_1, f_1) \subseteq M_2 = (V_2, f_2)$ if $supp(M_1) \subseteq supp(M_2)$ and for all $a \in V_1$, $f_1(a) \leq f_2(a)$. The union of two multisets is defined as $(M_1 \cup M_2) = (V_1 \cup V_2, f')$ where for all $a \in V_1 \cup V_2$, $f'(a) = f_1(a) + f_2(a)$, the difference is defined for $M_2 \subseteq M_1$ as $(M_1 - M_2) = (V_1 - V_2, f'')$ where $f''(a) = f_1(a) - f_2(a)$ for all $a \in V_1 - V_2$, and the intersection of two multisets is $(M_1 \cap M_2) = (V_1 \cap V_2, f''')$ where for $a \in V_1 \cap V_2$, $f'''(a) = min(f_1(a), f_2(a))$, $min(x, y)$ denoting the minimum of $x, y \in \mathbb{N}$. We say that M is empty, denoted by ϵ, if its support is empty, $supp(M) = \emptyset$.

A multiset M over the finite set of objects V can be represented as a string w over the alphabet V with $|w|_a = f(a)$, $a \in V$, and with ε representing the empty multiset ϵ. In the following we sometimes identify the finite multiset of objects $M = (V, f)$ with the word w over V representing M, thus we write $w \in V^\circ$, or sometimes we enumerate the elements of $w = a_1 \dots a_t \in V^\circ$ in double brackets (to distinguish from the usual set notation) as $\{\{a_1, \dots, a_t\}\}$.

Now we present the basic notions of membrane computing; the interested reader may find more detailed information on the theory of P systems in the monograph [10]. A P system is a structure of hierarchically embedded membranes, each having a label and enclosing a region containing a multiset of objects and possibly other membranes. The out-most membrane which is unique and usually labelled with 1, is called the skin membrane. The membrane structure is denoted by a sequence of matching parentheses where the matching pairs have the same label as the membranes they represent. If $x \in \{[_i,]_i \mid 1 \leq i \leq n\}^*$ is such a string of matching parentheses of length $2n$, denoting a structure where membrane i contains membrane j, then $x = x_1 \ [_i \ x_2 \ [_j \ x_3 \ ]_j \ x_4 \ ]_i \ x_5$ for some $x_k \in \{[_l,]_l \mid 1 \leq l \leq n,\ l \neq i, j\}^*,\ 1 \leq k \leq 5$. If membrane i contains membrane j, and there is no other membrane, k, such that k contains j and i contains k (x_2 and x_4 above are strings of matching parentheses themselves), then we say that membrane i is the parent membrane of j, denoted by $i = parent(j)$, and at the same time, membrane j is one of the child membranes of i.

By the contents of a region we mean the multiset of objects which is contained by the corresponding membrane excluding those objects which are contained by any of its child membranes.

The evolution of the contents of the regions of a P system is described by rules associated to the regions. Applying the rules synchronously in each region, the system performs a computation by passing from one configuration to another one. In the following we concentrate on communication rules called symport or antiport rules.

A symport rule is of the form (x, in) or $(x, out), x \in V^\circ$. If such a rule is present in a region i, then the objects of the multiset x must enter from the parent region or must leave to the parent region, $parent(i)$. An antiport rule is of the form $(x, in; y, out), x, y \in V^\circ$, in this case, objects of x enter from the parent region and in the same step, objects of y leave to the parent region. All types of these rules might be equipped with a promoter or inhibitor multiset, denoted as $(x, in)|_Z, (x, out)|_Z$, or $(x, in; y, out)|_Z, x, y \in V^\circ, Z \in \{z, \neg z \mid z \in V^\circ\}$, in which case they can only be applied if region i contains the objects of multiset z, or if $Z = \neg z$, then region i must not contain the elements of z. (For more on symport/antiport see [9], for the use of promoters see [8].)

The rules can be applied in the maximal parallel or in the sequential manner. When they are applied in the sequential manner, one rule is applied in each region in each derivation step, when they are applied in the parallel manner, as many rules are applied in each region as possible. See Definition 2 for the formal description of these modes.

The end of the computation is defined by halting: A P system halts when no more rules can be applied in any of the regions. In the case of P automata, however, we consider predefined accepting configurations called final states, by associating a finite set of multisets to each region. The P automaton accepts the input sequence when the contents of each region coincides with one element of these previously given finite sets of multisets.

The result of the computation can also be given in several ways, see [10] for more details. In the case of P automata, the result of the computation is an accepted multiset sequence, the sequence of multisets entering the skin membrane during a successful computation.

Now we present the formal definition of a P automaton.

Definition 1. A *P automaton* with n membranes is defined as $\Gamma = (V, \mu, (w_1, P_1, F_1), \ldots, (w_n, P_n, F_n))$ where $n \geq 1$, V is a finite alphabet of objects, μ is a membrane structure of n membranes with membrane 1 being the skin membrane, and for all $i, 1 \leq i \leq n$,

- $w_i \in V^\circ$ is the initial contents (state) of region i, that is, it is the finite multiset of all objects contained by region i,
- P_i is a finite set of communication rules associated to membrane i, they can be symport rules or antiport rules, with or without promoters or inhibitors, as above, and
- $F_i \subseteq V^\circ$ is a finite set of finite multisets over V called the set of final states of region i. If $F_i = \emptyset$, then all the states of membrane i are considered to be final.

To simplify the notations we denote symport and antiport rules with or without promoters/inhibitors as $(x, in; y, out)|_Z$, $x, y \in V^\circ, Z \in \{z, \neg z \mid z \in V^\circ\}$ where we also allow x, y, z to be the empty string. If $y = \varepsilon$ or $x = \varepsilon$, then the notation above denotes the symport rule $(x, in)|_Z$ or $(y, out)|_Z$, respectively, if $Z = \varepsilon$, then the rules above are without promoters or inhibitors.

The n-tuple of finite multisets of objects present in the n regions of the P automaton Γ describes a *configuration* of Γ; $(w_1, \ldots, w_n) \in (V^\circ)^n$ is the initial configuration.

Definition 2. The transition mapping of a P automaton is a partial mapping $\delta_X : V^\circ \times (V^\circ)^n \to 2^{(V^\circ)^n}$, with $X \in \{seq, par\}$ for sequential or for parallel rule application. These mappings are defined implicitly by the rules of the rule sets $P_i,\ 1 \leq i \leq n$. For a configuration $(u_1, \ldots, u_n)$,

$$(u'_1, \ldots, u'_n) \in \delta_X(u, (u_1, \ldots, u_n))$$

holds, that is, while reading the input $u \in V^\circ$ the automaton may enter the new configuration $(u'_1, \ldots, u'_n) \in (V^\circ)^n$, if there exist rules as follows.

– If $X = seq$, then for all $i, 1 \leq i \leq n$, there is a rule $(x_i, in; y_i, out)|_{Z_i} \in P_i$ with $z \subseteq u_i$ for $Z_i = z \in V^\circ$, and $z \cap u_i = \epsilon$ for $Z_i = \neg z, z \in V^\circ$, satisfying the conditions below, or

– if $X = par$, then for all $i, 1 \leq i \leq n$, there is a multiset of rules $R_i = \{\{r_{i,1}, \ldots, r_{i,m_i}\}\}$, where $r_{i,j} = (x_{i,j}, in; y_{i,j}, out)|_{Z_{i,j}} \in P_i$ with $z \subseteq u_i$ for $Z_{i,j} = z \in V^\circ$, and $z \cap u_i = \epsilon$ for $Z_{i,j} = \neg z, z \in V^\circ,\ 1 \leq j \leq m_i$, satisfying the conditions below, where $x_i,\ y_i$ denote the multisets $\bigcup_{1 \leq j \leq m_i} x_{i,j}$ and $\bigcup_{1 \leq j \leq m_i} y_{i,j}$, respectively. Furthermore, there is no $r \in P_j$, for any $j,\ 1 \leq j \leq n$, such that the rule multisets R'_i with $R'_i = R_i$ for $i \neq j$ and $R'_j = \{\{r\}\} \cup R_j$, also satisfy the conditions.

The conditions are given as

1. $x_1 = u$, and
2. $\bigcup_{parent(j)=i} x_j \cup y_i \subseteq u_i,\ 1 \leq i \leq n,$

and then the new configuration is obtained by

$$u'_i = u_i \cup x_i - y_i \cup \bigcup_{parent(j)=i} y_j - \bigcup_{parent(j)=i} x_j,\ 1 \leq i \leq n.$$

We define the sequence of multisets of objects accepted by the P automaton as an input sequence which is consumed by the skin membrane while the system reaches a final state, a configuration where for all j with $F_j \neq \emptyset$, the contents $u_j \in V^\circ$ of membrane j is "final", i.e., $u_j \in F_j$.

Note that in the case of parallel rule application, the set of multisets which may enter the system in one step is not necessarily bounded, thus, this type of automata may work with strings over infinite alphabets. In this paper however, we study languages over finite alphabets, so we apply a mapping to produce a finite set of symbols from a possibly infinite set of multisets, and in order not to "encode" the computational power in this mapping, we assume that it is computable by a linear space bounded Turing machine.

Definition 3. Let us extend δ_X to $\bar{\delta}_X, X \in \{seq, par\}$, a function mapping $(V^\circ)^*$, the sequences of finite multisets over V, and $(V^\circ)^n$, the configurations of Γ, to new configurations. We define $\bar{\delta}_X$ as

1. $\bar{\delta}_X(v, (u_1, \dots, u_n)) = \delta_X(v, (u_1, \dots, u_n)),\ v, u_i \in V^\circ$, $1 \leq i \leq n$, and
2. $\bar{\delta}_X((v_1)\dots(v_{s+1}), (u_1, \dots, u_n)) = \bigcup \delta_X(v_{s+1}, (u'_1, \dots, u'_n))$
   for all $(u'_1, \dots, u'_n) \in \bar{\delta}_X((v_1)\dots(v_s), (u_1, \dots, u_n))$, $v_j, u_i, u'_i \in V^\circ$,
   $1 \leq i \leq n,\ 1 \leq j \leq s+1$.

Note that we use brackets in the multiset sequence $(v_1)\dots(v_{s+1}) \in (V^\circ)^*$ in order to distinguish it from the multiset $v_1 \cup \dots \cup v_{s+1} \in V^\circ$.

Definition 4. Let Γ be a P automaton as above with initial configuration $(w_1, \dots, w_n)$ and let Σ be a finite alphabet. The *language accepted by* Γ in the sequential way of rule application, L_{seq}, or in the maximal parallel way of rule application, L_{par}, is

$$L_X(\Gamma) = \{f(v_1)\dots f(v_s) \in \Sigma^* \mid (u_1, \dots, u_n) \in \bar{\delta}_X((v_1)\dots(v_s), (w_1, \dots, w_n)) \text{ with } u_j \in F_j \text{ for all } j \text{ with } F_j \neq \emptyset,\ 1 \leq j \leq n,\ 1 \leq s\},$$

for $X \in \{seq, par\}$, and for a linear space computable mapping $f : V^\circ \longrightarrow \Sigma \cup \{\varepsilon\}$ with $f(x) = \varepsilon$ if and only if $x = \epsilon$. Let us denote the class of languages accepted by P automata with sequential or parallel rule application as $\mathcal{L}_X(PA)$, $X \in \{seq, par\}$.

3 The Power of P Automata

Now we consider the accepting power of P automata. We follow ideas from [5] and [6] in relating this power to well-known machine based complexity classes. There, among other similar models, the so-called symport/antiport P system acceptors are studied. These are accepting membrane systems similar to P automata, the main difference in the two models is the fact that the alphabet of symport/antiport acceptors is divided into a set of terminals and nonterminals. During the work of these systems both types of objects may leave or enter the membrane structure but only the objects which are terminal constitute the part of the input sequence which is accepted in a successful computation. Thus, the nonterminal objects are used to provide additional workspace for the computation.

This feature motivated the introduction of so-called $S(n)$ space bounded symport/antiport acceptors, systems where the total number of objects used in an accepting computation on a sequence of length n is bounded by a function $S(n)$. As shown in [5] and [6], a language L is accepted by an n^k space bounded symport/antiport acceptor, if and only if, it is accepted by a nondeterministic $\log n$ space bounded one-way Turing machine, or by a c^n space bounded symport/antiport acceptor, if and only if it is accepted by a one-way linear space bounded Turing machine, that is, if and only if it is context-sensitive.

Since in P automata the workspace is provided by the objects of the accepted or rejected input only, the maximal number of objects present inside the membrane structure during a computation is bounded by the length of the input sequence. Furthermore, even this "space" can only be used with a strong restriction since it becomes available step-by-step, as more and more symbols of the input are read. Thus, when looking for a Turing machine model corresponding to P automata, some restriction on the use of the available workspace is necessary.

Definition 5. A nondeterministic one-way Turing machine is *restricted* $S(n)$ *space bounded* if for every accepted input of length n, there is an accepting computation where the number of nonempty cells on the work-tape(s) is bounded by $S(d)$ where $d \leq n$, and d is the number of input tape cells already read, that is, the *distance* of the reading head from the left end of the one-way input tape.

Let $\mathcal{L}(1LOG)$, $\mathcal{L}(1LIN)$, $\mathcal{L}(\mathit{restricted}{-}1LOG)$, and $\mathcal{L}(\mathit{resticted}{-}1LIN)$ denote the class of languages accepted by one-way nondeterministic Turing machines with logarithmic space bound, linear space bound, restricted logarithmic space bound, and restricted linear space bound, respectively.

Let L denote the language

$$L = \{xy \mid x \in \{1, 2, \ldots, 9\}\{0, 1, \ldots, 9\}^*, y \in \{\#\}^+, \text{ with } val(x) = |y|\}$$

where $val(x)$ is the value of x as a decimal number.

As we shall see later, $\mathcal{L}(\mathit{restricted} - 1LIN) = \mathcal{L}(1LIN)$, but the class $\mathcal{L}(\mathit{restricted} - 1LOG)$ is strictly a subclass of $\mathcal{L}(1LOG)$ since L, the language defined above, is in the latter class but not in the former. Still, $\mathcal{L}(\mathit{restricted} - 1LOG)$ contains some very interesting languages, e.g., $\{a^n b^n c^n \mid n \geq 1\}$ and $\{a^{2^n} \mid n \geq 0\}$ are both in $\mathcal{L}(\mathit{restricted} - 1LOG)$, as they can be accepted by Turing machines capable of recording the distance of the reading head from the left-end of the one-way input tape which can be achieved in restricted logarithmic space.

Theorem 1.

$$\mathcal{L}_{seq}(PA) = \mathcal{L}(\mathit{restricted} - 1LOG) \text{ and } \mathcal{L}_{par}(PA) = \mathcal{L}(\mathit{restricted} - 1LIN).$$

Proof. First we prove the inclusions from left to right in both equations. Consider the P automaton $\Gamma = (V, \mu, (w_1, P_1, F_1), \ldots, (w_n, P_n, F_n)),\ n \geq 1$, with $L(\Gamma) \subseteq \Sigma^*$ where $f : V^\circ \longrightarrow \Sigma \cup \{\varepsilon\}$ is a linear space computable mapping with $f(x) = \varepsilon$ if and only if $x = \epsilon$. We show how to construct a one-way Turing machine M which simulates the work of Γ using restricted logarithmic space if Γ applies the rules sequentially, or restricted linear space if Γ applies the rules in the maximal parallel manner. Let $M = (k, \Sigma, A, Q, q_0, q_F, \delta_M)$ be a Turing machine with a one-way read only input tape where

- $k = (|V| \cdot n^2 + |V|)$ is the number of work-tapes,
- Σ is the finite input alphabet,
- $A = \{0, \ldots, 9\}$ is the work-tape alphabet,

- Q is the set of internal states, $q_0, q_F \in Q$ are the starting and the final states, and
- $\delta_M$is the transition function of M.

Let M have n work-tapes assigned to each region and symbol pair $(i, a) \in \{1, \ldots, n\} \times V$, and an additional tape for each symbol of V. Using the digits of the tape alphabet, $\{0, \ldots, 9\}$, M keeps track of the configurations of Γ by having an integer written on the first one of the work-tape n-tuple assigned to (i, a) denoting the number of a objects present in region i.

Let these configurations of M be denoted as

$$(q, w, \alpha_{1,1}, 0^{n-1}, \ldots, \alpha_{n,|V|}, 0^{n-1}, 0^{|V|})$$

where $q \in Q$ is the current state, $w \in \Sigma^*$ is the part of the input that is not yet read, $(\alpha_{i,j}, 0^{n-1}) \in (A^+)^n, 1 \leq i \leq n,\ 1 \leq j \leq |V|$, are the values written on the work-tape n-tuple corresponding to region i and symbol $a_j \in V$, the value of $\alpha_{i,j}$ denoting the number of such objects present in region i. For the sake of notational convenience, in the following we will use $\alpha_{i,j}$ to represent both the string of digits on the work-tape and the decimal value of this string.

Now let us consider the transitions of Γ. Let δ_Γ be δ_{seq} or δ_{par} as defined above for the case of sequential or maximal parallel way of rule application. The transition function of M is defined in such a way that if and only if

$$(u'_1, \ldots, u'_n) \in \delta_\Gamma(v, (u_1, \ldots, u_n)),$$

then

$$(q, w, \alpha_{1,1}, 0^{n-1}, \ldots, \alpha_{n,|V|}, 0^{n-1}, 0^{|V|}) \longrightarrow (q, w', \alpha'_{1,1}, 0^{n-1}, \ldots, \alpha'_{n,|V|}, 0^{n-1}, 0^{|V|})$$

is a possible transition in M where $\alpha_{i,j} = |u_i|_{a_j}$ and $\alpha'_{i,j} = |u'_i|_{a_j}$ for all $1 \leq i \leq n,\ 1 \leq j \leq |V|$, and if $f(v) = a$ then $w = aw'$, or if $v = \epsilon$, then $w = w'$. Note that this is possible because the finite set of rules can be encoded in the finite control, and all the information necessary to record a configuration of the P automaton is stored on the work-tapes. First, for each region and symbol pair, (i, a), M writes the number of a objects leaving from region i to region j to the jth tape of the work-tape n-tuple corresponding to (i, a), and also records the number of symbols entering from the environment using the $|V|$ additional work-tapes. Then in a final round, it creates the description of the new configuration by adding the appropriate values to the integers stored on the first tapes of each work-tape n-tuple, and using the collection of objects, $v \in V^\circ$, which enter from the environment, M computes $f(v) = a \in \Sigma$ and reads a from the input tape.

If Γ is a sequential P automaton, then this whole process can be realized in restricted logarithmic space since the number of cells used on the work-tapes is the logarithm of the number of objects present in the P system which is at most $c \cdot d$ where c is some constant and d is the number of nonempty multisets read by the P automaton, or in terms of the Turing machine, the distance of the reading head from the left end of the input tape. Furthermore, the function

f maps a finite domain to a finite set of values, so its computation does not require any additional space. If Γ works in the maximal parallel manner, then the computation of M requires restricted linear space because the number of symbols inside the P systems is at most c^d with c, d as above, so the integers describing the configurations can be represented by decimal numbers in restricted linear space. The values of $f(x) \in \Sigma \cup \{\varepsilon\}$ can also be computed inside the restricted linear space bound, since the cardinality of any $x \in V^\circ$ for which the computation is needed is at most c^d, and the computation of $f(x)$ itself uses linear space measured in the size of the argument, so it is still restricted linear.

The transition function of M should also enable an initialization phase,

$$(q_0, w, \varepsilon, \ldots, \varepsilon) \longrightarrow (q, w, \alpha_{1,1}, 0^{n-1}, \ldots, \alpha_{n,m}, 0^{n-1}, 0^{|V|})$$

where $\alpha_{i,j} = |w_i|_{a_j},\ 1 \leq i \leq n,\ 1 \leq j \leq |V|$.

The input is accepted by M if and only if it is accepted by Γ, that is,

$$(q, \varepsilon, \alpha_{1,1}, 0^{n-1}, \ldots, \alpha_{n,m}, 0^{n-1}, 0^{|V|}) \longrightarrow \\ (q_F, \varepsilon, \alpha_{1,1}, 0^{n-1}, \ldots, \alpha_{n,m}, 0^{n-1}, 0^{|V|})$$

where for each $F_i \neq \emptyset$, there is an $u_i \in F_i$ such that for all $a_j \in V,\ \alpha_{i,j} = |u_i|_{a_j}$. The precise construction of M is left to the reader.

Now we prove the inclusions from right to left. To do this we need the notion of a two-counter automaton. A *two-counter machine* is an automaton with a one-way read only input tape and two counters capable of storing any non-negative integer. Formally it can be given as $M = (\Sigma, Q, q_0, q_F, \delta_M)$ where Σ is an input alphabet, Q is a set of internal states containing the initial and accepting states $q_0, q_F \in Q$ respectively, and δ_M is a transition function which maps the quadruple of state, input symbol, and zero or non-zero counter contents to the triple of a new state and two instructions to increment, unchange, or decrement the counters. As the work-tapes of any Turing machine can be simulated with two counters, two-counter machines accept the class of recursively enumerable languages (see [2]).

However, if the sum of the counter contents is bounded, that is, the two-counter automaton has limited workspace, then its power is decreased. If we define $S(n)$ space bounded two-counter machines as $S(n)$ being the bound on the sum of the counter contents during any accepting computation on an input of length n, then we obtain a model equivalent to $\log S(n)$ space bounded one-way Turing machines because an integer i stored in a counter can be written on the work-tapes using $\log i$ tape cells. We may also introduce the restricted $S(n)$ space bounded variant exactly as above, in which case we obtain a machine equivalent to restricted $\log S(n)$ space bounded one-way Turing machines.

Consider now a two-counter machine M. If x is an element of the domain of δ_M, then a transition is given by the pair $(x, \delta_M(x))$. Let these pairs be labelled by elements of the finite set of labels $TRANS$.

Let $\Gamma = (V, \mu, (w_1, P_1, \emptyset), (w_2, P_2, F_2), (w_3, P_3, \emptyset), (w_4, P_4, \emptyset))$ be a P automaton with the membrane structure $\mu = [_1\ [_2\ ]_2\ [_3\ ]_3\ [_4\ ]_4\ ]_1$. Inside the skin membrane, it has a controlling region, region 2, for storing and manipulating symbols corresponding to the states of M, and a pair of membranes, 3 and 4, for maintaining the values of the two counters.

Let $V = \Sigma \cup \{\langle q \rangle, \langle t \rangle, \langle t \rangle_a \mid q \in Q,\ t \in \mathit{TRANS},\ a \in \Sigma\}$. The symbols of $V - \Sigma$ govern the work of Γ. The presence of $\langle q \rangle \in V$ in the skin membrane indicates that Γ simulates a configuration of the two-counter machine when it is in state $q \in Q$. While simulating a transition from state q to state q' labelled by $t \in \mathit{TRANS}$, Γ needs extra steps for reading the input and manipulating the symbols which keep track of the counter values. During these steps the skin membrane contains one of the symbols $\langle t \rangle$, $\langle t \rangle_a$, for some $a \in \Sigma$, and when the simulation of the transition is complete, $\langle q' \rangle \in V$ appears in the skin membrane.

The simulation of M starts in the initial state with $w_1 = \langle q_0 \rangle\langle q_0 \rangle\langle q_0 \rangle a$, for some $a \in \Sigma$, and for $2 \leq i \leq 4$, $w_i = \{\{\ \langle q \rangle, \langle t \rangle, \langle t \rangle_a \mid q \in Q,\ q \neq q_0,\ t \in \mathit{TRANS},\ \ a \in \Sigma\ \}\}$. The rule sets belonging to the regions are as follows. Let

$$\begin{aligned} P_1 = \{&(\epsilon, in)|_{\langle q \rangle\langle q \rangle\langle q \rangle},\ (\epsilon, in)|_{\langle t \rangle_a\langle t \rangle_a\langle t \rangle_a} \mid q \in Q,\ t \in \mathit{TRANS},\ a \in \Sigma\} \cup \\ &\{(x^k, in; y, out)|_{\langle t \rangle\langle t \rangle\langle t \rangle} \mid x \in \Sigma,\ y \in \Sigma,\ t \in \mathit{TRANS},\ x \text{ is read} \\ &\quad \text{during the transition } t\} \cup \\ &\{(\epsilon, in)|_{\langle t \rangle\langle t \rangle\langle t \rangle} \mid t \in \mathit{TRANS},\ \varepsilon \text{ is read during the transition } t\}, \end{aligned}$$

and $k \geq 1$ is a suitable constant. In the case of sequential rule application, if transition t is simulated, as indicated by the presence of $\langle t \rangle$, then k copies of the corresponding input symbol are read into the skin membrane, and one other symbol is sent out in the first simulating step. If the rules are applied in the maximal parallel manner, then after the first simulating step, the number of symbols in the skin region is k times as much, as the number that was already present.

Before the simulation starts, that is, when a state symbol $\langle q \rangle$ is present, or in the later simulating steps, when the symbols $\langle t \rangle_a$, for some $a \in \Sigma$, are present, then nothing is read from the input.

In what follows, let us assume that $\Sigma = \{a_1, \ldots, a_m\}$. Now let

$$\begin{aligned} P_2 = \{&(\langle q \rangle, in; \langle t \rangle, out), (\langle t \rangle, in; \langle t \rangle_{a_1}, out), (\langle t \rangle_{a_i}, in; \langle t \rangle_{a_{i+1}}, out), \\ &(\langle t \rangle_{a_m}, in; \langle q' \rangle, out) \mid 1 \leq i \leq m-1,\ t \in \mathit{TRANS} \text{ is a} \\ &\text{transition from } q \text{ to } q'\}, \end{aligned}$$

and let $F_2 = \{\ \{\{\ \langle q \rangle, \langle t \rangle, \langle t \rangle_a \mid q \in Q,\ q \neq q_F,\ t \in \mathit{TRANS},\ \ a \in \Sigma\ \}\}\ \}$. The second region is responsible for keeping track of the simulated states and transitions. The simulation is finished if q_F, the final state is exported from this region to the first one as indicated by F_2 above. Now for $3 \leq i \leq 4$, let

$$\begin{aligned} P_i = \{&(\langle q \rangle, in; \langle t \rangle x, out), (\langle t \rangle, in; \langle t \rangle_{a_1}, out), (\langle t \rangle_{a_j}, in; \langle t \rangle_{a_{j+1}}, out), \\ &(\langle t \rangle_{a_m}, in; \langle q' \rangle, out) \mid x \in \Sigma,\ 1 \leq j \leq m-1,\ t \in \mathit{TRANS} \\ &\text{is a transition from } q \text{ to } q' \text{ decreasing the } i\text{th counter }\} \cup \\ &\{(\langle q \rangle, in; \langle t \rangle, out), (\langle t \rangle, in; \langle t \rangle_{a_1}, out)|_{Z_1}, (\langle t \rangle_{a_j}, in; \langle t \rangle_{a_{j+1}}, out)|_{Z_{j+1}}, \\ &(\langle t \rangle_{a_m} x, in; \langle q' \rangle, out) \mid 1 \leq j \leq m-1,\ t \in \mathit{TRANS} \text{ is a transition} \end{aligned}$$

from q to q', and
$x \in \Sigma$ if the $(i-2)$th counter is increased during t, or
$x = \varepsilon$ if the $(i-2)$th counter is not changed during t, or
$Z_j = \varepsilon$ for all $a_j \in \Sigma$ if the $(i-2)$th counter can be nonempty before t, or
$Z_j = \neg a_j$ for all $a_j \in \Sigma$ if the $(i-2)$th counter must be empty before $t\}$.

Region 3 and 4 keep track of the values stored in the counters of M by the help of the rules above. Together with moving the transition symbols $\langle t\rangle_a$, for some $a \in \Sigma$, they import and export the symbols originating from the input as necessary to maintain the correct counter contents, the emptiness of the counters are checked by the use of the forbidding promoters.

In the case of sequential rule application, the possible inputs are the multisets containing the elements of Σ in k copies, so we can use the mapping $f_{seq}(\{\{x^k\}\}) = x$, $x \in \Sigma$ to map V° to Σ. In the parallel case, the input multisets are of the form x^{ik} for some $x \in \Sigma$, $i \geq 1$, so we might use $f_{par}(x^{ik}) = x$, for all $i \in \mathbb{N}$, to produce the string corresponding to an input sequence.

Now, if M is restricted $S(n)$ space bounded with $S(n) = c \cdot n$ for a constant c, then it can be simulated by the P automaton Γ with sequential rule application, since if $k \geq c+1$, then the number of objects (the bound on the sum of the counter values) is $d(k-1)+1 \geq d \cdot c$ where d is the number of already read nonempty multisets. If M is restricted $S(n)$ space bounded with $S(n) = c^n$ for a constant c, then it can be simulated by Γ with parallel rule application, since if $k \geq 2 \cdot c$, then the amount of imported objects during the computation is sufficient to make sure that in each step the number of available objects which can be used to keep track of the counter values is c^d, where d is the number of nonempty multisets read. Since restricted $S(n)$ space bounded two-counter machines are equivalent to restricted $\log S(n)$ space bounded one-way Turing machines, our statement is proved. □

Based on this theorem, we can show that the class of languages accepted by P automata in the sequential way is strictly included in the class of languages accepted by logarithmic space bounded one-way Turing machines, that is, in $\mathcal{L}(1LOG)$.

Theorem 2. $\mathcal{L}_{seq}(PA) \subset \mathcal{L}(1LOG)$.

Proof. To prove our statement we show that $\mathcal{L}(restricted - 1LOG)$ is strictly included in $\mathcal{L}(1LOG)$. The inclusion is obvious, so it is enough to show that the difference of the two classes is nonempty. Consider the language L as defined above in Section 2. It is clear that L can be accepted by a one-way Turing machine using logarithmic space: First the initial part, $x \in \{1,\ldots,9\}\{0,\ldots,9\}^*$, of the input is copied to a work-tape then the number of # symbols are checked against this integer by reading further and decreasing the stored integer by one in each step.

To see that L cannot be accepted by a one-way restricted logarithmic space bounded Turing machine, suppose that M is such a machine accepting L. Without loss of generality, assume that M has only one work-tape. Then there is a constant c (which only depends on the specification of M) such that for every k, after reading k input symbols, M uses at most $c \cdot \log k$ cells on the work-tape. (All logarithms are base 10.) Also, the number of possible configurations on a work-tape of at most $c \cdot \log k$ cells (a configuration is a triple consisting of the state, work-tape contents, work-tape head position) is at most $d^{\log k} = k^{\log d} \leq k^e$ for some constants d and e. Now for any given k, consider the set of all strings of the form: $w\#^{val(w)}$, where $|w| = k$. Clearly, $k \leq g \cdot \log n$ for some constant g, where $n = |w\#^{val(w)}|$. Since the machine is one-way restricted logarithmic space bounded, there exist an n (and therefore k) big enough and two inputs $w\#^{val(w)}$ and $w'\#^{val(w')}$ with $|w| = |w'| = k$ and $w \neq w'$ for which M will be in the same configuration after reading w and after reading w'. This is because M can be in one of at most $k^e \leq (g \cdot \log n)^e$ configurations after reading a string of length k. Now there are $10^k \geq h \cdot n$ strings w whose lengths are k for some positive constant h. If we choose n big enough, $(g \cdot \log n)^e < h \cdot n$. It follows that the machine when given the string $w'\#^{val(w)}$ will also accept. This is a contradiction since $w'\#^{val(w)}$ is not in L. $\square$

It is interesting to look at the closure properties of $\mathcal{L}_{seq}(PA)$ which, as shown in Theorem 1 is identical to $\mathcal{L}(restricted - 1LOG)$.

Theorem 3. *$\mathcal{L}_{seq}(PA)$ is closed under union, intersection, concatenation, Kleene * and $^+$, inverse homomorphism, and ε-free homomorphism. It is not closed under (unrestricted) homomorphism, reversal, and complementation.*

Proof. It is straightforward to verify that $\mathcal{L}_{seq}(PA) = \mathcal{L}(restricted - 1LOG)$ is closed under union, intersection, concatenation, Kleene * and $^+$, inverse homomorphism, and ε-free homomorphism. By using $L' = \{xyz \mid x \in \{\&\}^+, y \in \{1, \ldots, 9\}\{0, \ldots 9\}^*, z \in \{\#\}^+, |x| = |z| = val(y)\}$, a variant of the language defined above, it is easy to see that $\mathcal{L}(restricted - 1LOG)$ is not closed under unrestricted homomorphism, since erasing the symbol & produces L from L'. It is also not closed under reversal since the reverse of L is clearly in $\mathcal{L}(restricted - 1LOG)$, but as we have seen, L is not. To see that $\mathcal{L}(restricted - 1LOG)$ is not closed under complementation, consider $\bar{L}$, the complement of L. $\bar{L}$ can be accepted by a one-way Turing machine M' using restricted logarithmic space, as follows. Inputs that do not have the form xy with $x \in \{1, \ldots, 9\}\{0, \ldots, 9\}^*$, $y \in \{\#\}^+$ can easily be accepted by M'. For an input of the form xy, M' needs to check that $val(x) \neq |y|$. To do this, M' scans the segment x using its work-tape as a counter (in base 10) to record the position of the input head as it scans x. At some point, nondeterministically chosen, M' stops incrementing the counter and remembers the symbol d under the input head. Note that at this point, the counter has value $\log i$ and d is the i-th symbol of x. M' then scans the segment y while recording the length of y on another work-tape (again use as a counter in base 10). Let w be the count after processing y. M' then checks that the symbol in position i of w is not equal to d. Clearly M' accepts $\bar{L}$. Since L is not in $\mathcal{L}(restricted-1LOG)$, it follows that $\mathcal{L}(restricted-1LOG)$ is not closed under complementation. □

Now consider the *deterministic* version of a one-way restricted logarithmic space bounded Turing machine. It is easy to show that $\mathcal{L}(restricted{-}1DLOG)$ is closed under complementation. Hence, from Theorem 2 we have $\mathcal{L}(restricted{-}1LOG) - \mathcal{L}(restricted-1DLOG) \neq \emptyset$. This is an interesting example of nondeterminism being better than determinism for a restricted type of space-bounded Turing machine.

Unlike in the case of the logarithmic space bound, the restricted use of linear space does not influence the power of a linear space bounded Turing machine. To see this, consider the machine which first copies its input to an additional work-tape, then works with it as with the input tape, and with the rest of its work-tapes exactly as before. This machine uses restricted linear space, and it clearly accepts the same set of input words as before. Thus, since linearly bounded Turing machines characterize the class of context-sensitive languages, we have:

Theorem 4. $\mathcal{L}_{par}(PA) = \mathcal{L}(1LIN) = \mathcal{L}(CS)$.

References

1. Csuhaj-Varjú, E., Vaszil, Gy.: *P* Automata. In: Păun, Gh., Zandron, C. (eds.): Pre-Proceedings of the Workshop on Membrane Computing WMC-CdeA 2002, Curtea de Argeş, Romania, August 19-23, 2002. Pub. No. 1 of MolCoNet-IST-2001-32008 (2002) 177-192, and also in
   Păun, Gh., Rozenberg, G., Salomaa, A., Zandron, C. (eds.): Membrane Computing. Lecture Notes in Computer Science, Vol. 2597. Springer, Berlin (2003) 219-233
2. Fischer, P. C.: Turing Machines with Restricted Memory Access. Information and Control **9** (1966) 364-379
3. Freund, R., Martín-Vide, C., Obtułowicz, A., Păun, Gh.: On Three Classes of Automata-like P Systems. In: Ésik, Z., Fülöp, Z. (eds.): Developments in Language Theory. 7th International Conference, DLT 2003, Szeged, Hungary, July 2003. Proceedings. Lecture Notes in Computer Science, Vol. 2710. Springer, Berlin (2003) 292-303
4. Freund, R., Oswald, M.: A Short Note on Analysing P Systems. Bulletin of the EATCS **78** (October 2002) 231-236
5. Ibarra, O. H.: On the Computational Complexity of Membrane Systems. To appear in Theoretical Computer Science C
6. Ibarra, O. H.: The Number of Membranes Matters. In: Alhazov, A., Martín-Vide, C., Păun, Gh. (eds.): Workshop on Membrane Computing, WMC-2003, Tarragona, July 17-22, 2003. Technical Report 28/03 of the Research Group on Mathematical Linguistics, Rovira i Virgili University, Tarragona, Spain (2003) 273-285
7. Madhu, M., Krithivasan, K.: On a Class of P Automata. Submitted
8. Martín-Vide, C., Păun, A., Păun, Gh.: On the Power of P Systems with Symport Rules. Journal of Universal Computer Science **8**(2) (2002) 317-331

9. Păun, A., Păun, Gh.: The Power of Communication: P Systems with Symport/Antiport. New Generation Computing **20**(3) (2002) 295-306
10. Păun, Gh.: Computing with Membranes: An Introduction. Springer, Berlin, (2002)
11. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages. Springer-Verlag, Berlin, vol. 1-3, (1997)

A Weighted Insertion-Deletion Stacked Pair Thermodynamic Metric for DNA Codes

Arkadii G. D'yachkov[1], Anthony J. Macula[2,⋆], Wendy K. Pogozelski[3,⋆⋆], Thomas E. Renz[2], Vyacheslav V. Rykov[4], and David C. Torney[5]

[1] Dept. of Probability Theory, Moscow State Univ., Moscow 119899, Russia
dyachkov@artist.math.msu.su

[2] Air Force Research Lab, IFTC, Rome Research Site, Rome NY 13441
thomas.renz@rl.af.mil, macula@geneseo.edu

[3] Department of Chemistry, SUNY Geneseo, Geneseo NY 14454
pogozels@geneseo.edu

[4] Dept. of Mathematics, University of Nebraska-Omaha, Omaha NE, 68182
vrykov@mail.unomaha

[5] Theoretical Biology and Biophys., Los Alamos Natl. Lab, Los Alamos NM 87545
dct@lanl.gov.edu

Abstract. Thermodynamic distance functions are important components in the construction of DNA codes and DNA codewords are structural and information building blocks in biomolecular computing and other biotechnical applications that employ DNA hybridization assays. We introduce new metrics for DNA code design that capture key aspects of the nearest neighbor thermodynamic model for hybridized DNA duplexes. One version of our metric gives the maximum number of stacked pairs of hydrogen bonded nucleotide base pairs that can be present in any secondary structure in a hybridized DNA duplex without pseudoknots. We introduce the concept of (t-gap) block isomorphic subsequences to describe new string metrics that are similar to the weighted Levenshtein insertion-deletion metric. We show how our new distances can be calculated by a generalization of the folklore longest common subsequence dynamic programming algorithm. We give a Varshamov-Gilbert like lower bound on the size of some of codes using our distance functions as constraints. We also discuss software implementation of our DNA code design methods.

⋆ Corresponding author and supported by AFOSR F30602-03-C-0059.
⋆⋆ Partially supported by AFOSR-36062-2-0086.

1 Introduction

Biomolecular computing often requires oligonucleotides that do not produce erroneous cross-hybridizations and synthetic DNA is proposed for use as an information storage media and structural material in nanotechnology. There is a need to efficiently create large collections of non-crosshybridizing oligonucleotides and the process of designing sets of non-crosshybridizing oligonucleotides has come to be known as *DNA code (word) design.* Both combinatorial and biological methods have been suggested as a means by which DNA codes can be found and programs exist that generate DNA codes. See [1], [2], [3], [4], [7], [8], [14], [16], [22] and the many references contained within.

In this paper, we introduce new metrics for DNA code design that capture key aspects of the nearest neighbor thermodynamic model for hybridized DNA duplexes. Our thermodynamically weighted distance functions are metrics in the rigorous mathematical sense. To construct our thermodynamic distance functions, in Section 2, we introduce the concept of *(t-gap) block isomorphic subsequences* and use it to describe new abstract weighted string metrics that are similar to the weighted *Levenshtein insertion-deletion metric.* In Section 4, we show how our new distances can be calculated by a generalization of the folklore dynamic programming algorithm for the longest common subsequence. In Theorems 1 and 2, we give a Varshamov-Gilbert like lower bound on the size of some of codes using these distance functions as constraints. Our method is a generalization of the ideas in [3], [5], [9], [10] where *all* potential secondary structures (i.e., common subsequences) in cross-hybridized duplexes are considered.

In this paper, all variables are nonnegative integers unless otherwise stated. $[n]$ denotes the set $\{0,...,n-1\}$ and (n) denotes the sequence $1,2,...,n$. Given two sequences α and β, we write $\alpha \prec \beta$ if and only if α is a subsequence of β. The length of sequence α is denoted by $|\alpha|$. We call $\alpha \prec (n)$ a *string* if and only if it is a subsequence of consecutive integers, e.g., $\alpha = i, i+1, ..., i+k$. For $a \leq b$, we use the notation $[a,b]$ for the string of integers between and including a and b. If $a = b$, we sometimes write $[a]$ for $[a,b]$. When we write $\sigma = [a_1,b_1],[a_2,b_2],...,[a_i,b_i],...[a_k,b_k]$ where $a_i \leq b_i < a_{i+1}$, we mean $\sigma = a_1, a_1+1, ..., b_1, a_2, a_2+1, ...b_2, ..., a_i, a_i+1, ...b_i, ...a_k, a_k+1, ...b_k$. For $\sigma \prec (n)$, $\tau \prec (m)$ with $|\sigma| \leq |\tau|$, we write $f : \sigma \to \tau$ to indicate an *increasing function* $f : \{i : i \in \sigma\} \to \{i : i \in \tau\}$. Given $\sigma = i_1, i_2, ..., i_k$ and $f : \sigma \to \tau$, we define $f(\sigma) \equiv f(i_1), f(i_2), ..., f(i_k)$. If $|\sigma| = |\tau|$, then $f : \sigma \to \tau$ is unique. We let $[q]^n$denote the set of sequences of length n with entries in $[q]$. For $x = x_1, ..., x_n$ with $x \in [q]^n$ and $\sigma = i_1, i_2, ..., i_k$ where $\sigma \prec (n)$, we let $x_\sigma \prec x$ be the subsequence $x_{i_1}, x_{i_2} ..., x_{i_k}$. Given a non-negative real-valued function, Ω, on $[q]$, we define $\|x_\sigma\|_\Omega \equiv \sum\limits_{i \in \sigma} \Omega(x_i)$.

2 Block Isomorphic Subsequences

Definition 1. *For $\sigma \prec (n)$, a substring $\beta \prec \sigma$ is called a block of σ if β is not subsequence of any substring α of σ with $\beta \neq \alpha$. A subsequence $x_\beta \prec x_\sigma$*

is called a block of x_σ if β is a block of σ. For $\sigma \prec (n)$, let $(\beta_i(\sigma))$ be the sequence of blocks of σ, where each element of $\beta_i(\sigma)$ is less than every element of $\beta_{i+1}(\sigma)$. Let $b_i(\sigma)$ and $B_i(\sigma)$ be the left and right endpoints of $\beta_i(\sigma)$. When the context is clear, we just write b_i, B_i and β_i respectively. When we write $\sigma = ([b_i, B_i]) = (\beta_i)$, we say that it is the block representation of σ. When we write $x_\sigma = x_{(\beta_i)}$ we say that it is the block representation of x_σ.

Block representations are unique. Let $x \in [4]^{14}$ and let $\sigma \prec (14)$. Suppose $x = 2,0,1,2,2,3,0,3,2,0,0,1,3,2$ and $\sigma = 2,3,4,7,9,10,13,14$. Then $x_\sigma = 0,1,2,0,2,0,3,2$ and the block representations are: $\sigma = [2,4]\,,\ [7,7]\,,\ [9,10]\,,\ [13,14]$ and $x_\sigma = x_{[2,4],[7,7],[9,10],[13,14]}$.

Definition 2. *For $2 \le t \le n-1$, we define $G_t(n) \equiv \{\sigma \prec (n) : b_{i+1}(\sigma) - B_i(\sigma) \ge t\}$. We call $G_t(n)$ the set of t-gap sequences of (n).*

Note that $\sigma \prec (n) \Rightarrow \sigma \in G_2(n)$ and $2 \le t_1 \le t_2 \le n-1 \Rightarrow G_{t_2}(n) \subseteq G_{t_1}(n)$.

Example 1. Let $x \in [4]^{14}$ and let $\sigma_i \prec (14)$. Then: $\sigma_1 = [1,4]$, $[7,7]$, $[9,10]$, $[12,14] \in G_2(n)$; $\sigma_2 = [1,4]$, $[7,7]$, $[10,10]$, $[13,14] \in G_3(n)$; $\sigma_3 = [1,4]$, $[13,14] \in G_4(n)$.

Definition 3. *Let $\sigma \prec (n)$, $\tau \prec (m)$ with $|\sigma| = |\tau|$. Let $f : \sigma \to \tau$ be unique. We say that σ and τ are block isomorphic and write $\sigma \cong \tau$ if: $\alpha \prec \sigma$ is a string $\Longleftrightarrow$ $f(\alpha) \prec \tau$ is a string. For $x \in [q]^n$, $y \in [q]^m$ we say that x_σ and y_τ are block isomorphic, denoted by $x_\sigma \cong y_\tau$, if $x_\sigma = y_\tau$ and $\sigma \cong \tau$.*

Proposition 1. *[12] Suppose $\sigma \cong \tau$ and $f : \sigma \to \tau$ is unique. Then $\beta \prec \sigma$ is a block in σ if and only if $f(\beta)$ is a block in τ.*

Definition 4. *Let $2 \le t \le \min(n,m) - 1$ and suppose $\sigma \prec (n)$, $\tau \prec (m)$ with $|\sigma| = |\tau|$. We say that σ and τ are t-gap block isomorphic and write $\sigma \underset{t}{\cong} \tau$ if and only if $\sigma \in G_t(n)$, $\tau \in G_t(m)$ and $\sigma \cong \tau$. For $x \in [q]^n$, $y \in [q]^m$,we say that x_σ and y_τ are t-gap block isomorphic, denoted by $x_\sigma \underset{t}{\cong} y_\tau$, if and only if $x_\sigma = y_\tau$ and $\sigma \underset{t}{\cong} \tau$. We say that x and y have a common t-gap block isomorphic subsequence if and only if there are $\sigma \prec (n)$, $\tau \prec (m)$ with $x_\sigma \underset{t}{\cong} y_\tau$. Note that $\sigma \cong \tau \Leftrightarrow \sigma \underset{2}{\cong} \tau$ and $\sigma \underset{t_2}{\cong} \tau \Rightarrow \sigma \underset{t_1}{\cong} \tau$ when $t_1 \le t_2$. So we just write $\sigma \cong \tau$ and $x_\sigma \cong y_\tau$ to denote $\sigma \underset{2}{\cong} \tau$ and $x_\sigma \underset{2}{\cong} y_\tau$ respectively.*

Example 2. Let $x, y, z, w \in [4]^{13}$ and $\sigma_i \prec (13)$ with x=1, 1, 2, 0, 2, 3, 3, 0, 1, 1, 2, 0, 1; y=2, 0, 2, 3, 3, 0, 1, 1, 1, 1, 2, 0, 3; z=3, 2, 0, 2, 1, 1, 1, 1, 1, 0, 3, 2, 0; w=1, 1, 1, 2, 0, 3, 2, 0, 2, 0, 3, 3, 3. Let σ_1 =[3,5], [8,8], [11,12]; σ_2 =[1,3], [6,6], [11,12]; σ_3 =[2,4], [10,10], [12,13]; σ_4 =[4,5], [7,10]. Then $x_{\sigma_1} = y_{\sigma_2} = z_{\sigma_3} = w_{\sigma_4} = 2,0,2,0,2,0$. Since $\sigma_1 \cong \sigma_2 \cong \sigma_3 \ncong \sigma_4$, we have that $x_{\sigma_1} \cong y_{\sigma_2} \cong z_{\sigma_3} \ncong w_{\sigma_4}$. Since $\sigma_1, \sigma_2 \in G_3(n)$ and $\sigma_3 \notin G_3(n)$, we have that $x_{\sigma_1} \underset{3}{\cong} y_{\sigma_2} \underset{3}{\ncong} z_{\sigma_3}$.

3 Block Insertion-Deletion Codes

Definition 5. *For $x, y \in [q]^n$, we define: $\rho_{\Omega,q}(x,y) \equiv \max\{\|z\|_{\Omega} : z \prec x$ and $z \prec y\}$ and $L_{\Omega,q}(x,y) \equiv \min(\|x\|_{\Omega}, \|y\|_{\Omega}) - \rho_{\Omega}(x,y)$. We say that $\rho_{\Omega}(x,y)$ is the maximum weight of a common subsequence to x and y and $L_{\Omega}(x,y)$ is called the weighted Levenshtein insertion-deletion distance. $L_{\Omega}(x,y)$ is a metric. When $\|x_\sigma\|_{\Omega} = |x_\sigma|$, we write $L(x,y)$ for $L_{\Omega}(x,y)$.*

Note that metrics similar to $L_{\Omega}(x,y)$ are used in DNA sequence analysis. See [24] and [15]. The proof that $L_{\Omega}(x,y)$ is a metric is a modification of an argument in [17] where it is indicated that $L^*(x,y)$ is a metric when $L^*_{\Omega}(x,y) \equiv \|x\|_{\Omega} + \|y\|_{\Omega} - 2\rho_{\Omega}(x,y)$. The following definition is analogous to that in Definition 5.

Definition 6. *For $2 \le t \le n-1$ and $x, y \in [q]^n$. We define:*

$$\phi^t_{\Omega,q}(x,y) \equiv \max\{\|x_\sigma\|_{\Omega} : x_\sigma \underset{t}{\cong} y_\tau\}. \tag{1}$$

$$\Phi^t_{\Omega,q}(x,y) \equiv \min(\|x\|_{\Omega}, \|y\|_{\Omega}) - \phi^t_{\Omega,q}(x,y). \tag{2}$$

When the context for q is clear, we simply write $\Phi^t_{\Omega}(x,y)$ and $\phi^t_{\Omega}(x,y)$. We say that $\phi^t_{\Omega}(x,y)$ is the weight of the longest common t-gap block subsequence of x and y. When $\|x_\sigma\|_{\Omega} = |x_\sigma|$, we write $\Phi^t(x,y)$ and $\phi^t(x,y)$ for $\Phi^t_{\Omega}(x,y)$ and $\phi^t_{\Omega}(x,y)$ respectively. For $t = 1$, we define $\phi^1_{\Omega}(x,y) \equiv L_{\Omega}(x,y)$ and $\phi^1(x,y) \equiv L(x,y)$.

Proposition 2. *[12] $\Phi^t_{\Omega}(x,y)$ is a metric on $[q]^n$.*

Definition 7. *A t-gap block insertion-deletion q-ary code of weighted Ω distance d is a subset C, of $[q]^n$, such that: $x \ne y \in C \Longrightarrow \Phi^t_{\Omega}(x,y) \ge d$.*

Theorem 1. *[12] Let $\|x_\sigma\|_{\Omega} = |x_\sigma|$. In $[q]^n$, there is a 2-gap block insertion-deletion C of $d = n-k$ with*

$$|C| \ge q^k \left(\sum_{j=1}^{k} \binom{k-1}{j-1} \binom{n-k+1}{j}^2 \right)^{-1}.$$

4 Computing $\phi^t_{\Omega}(x,y)$

For $1 \le m < n$, consider the string $[m+1, n]$. For $x, y \in [q]^n$, let suf(x,y) be the length of the longest common suffix between x and y. Then $suf(x,y) = 0$ if $x_n \ne y_n$ and $suf(x,y) = n-m$ if $x_{[m+1,n]} = y_{[m+1,n]}$ and $x_m \ne y_m$. For $x \in [q]^n$, we have that $x_{[1,i]}$ is the first i entries of x and $x_{[1,n]} = x$. Proposition 3 gives a dynamic programming method of computing $\phi^t_{\Omega}(x,y)$ that is a generalization of the folklore algorithm for the longest common subsequence.

Proposition 3. *[12] Let $1 \le t \le n-1$. For $x, y \in [q]^n$ and $t < i, j \le n$, define $M^t_{\Omega,i,j} \equiv \phi^t_\Omega(x_{[1,i]}, y_{[1,j]})$. Let $\omega(r) \equiv \left\| x_{[n-r+1,n]} \right\|_\Omega$ and $suf(x,y) = k$. Define $D^t_{\Omega,i,j} \equiv \max\{\omega(r) + M^t_{\Omega,i-r-t+1,j-r-t+1} : 1 \le r \le k\}\}$ if $k \ge 1$ and $D^t_{\Omega,i,j} \equiv 0$ if $k = 0$. Then*

$$M^t_{\Omega,i,j} = \phi^t_\Omega(x_{[1,i]}, y_{[1,j]}) = \max\{M^t_{\Omega,i-1,j}, M^t_{\Omega,i,j-1}, D^t_{\Omega,i,j}\}. \tag{3}$$

When either i or j is less than or equal to t, the initial conditions needed for the computation of $\phi^t_\Omega(x,y)$ are $\phi^t_\Omega(x_{[1,i]}, y_{[1,j]}) = \left\| x_{[1,i]} \right\|_\Omega$ if and only if $x_{[1,i]}$ is a substring of $y_{[1,j]}$.

Example 3. Let $x = 0,1,2,3,1,3,0,1$ and $y = 3,0,1,3,2,0,3,1$. Figure 1 and Figure 2 are M^2 and M^3 where $\|x_\sigma\| = |\sigma|$. Figure 3 and Figure 4 are M^2_Ω and M^3_Ω where $\|x_\sigma\|_\Omega \equiv \sum\limits_{i\in\sigma} (x_i + 1)$. Below each figure, we give an example of t-gap isomorphic subsequences with $\|x_\sigma\| = \|y_\tau\| = \varphi^t(x,y)$ $(\varphi^t(x,y).)$

5 Sequences of t-Strings

In this section, we apply the results Sections 2 and 3 to sequences of strings of length t (with particular attention to $t = 2$) that naturally arise from $x \in [q^n]$. The goal is to then apply these results to the modeling of DNA hybridization distances. This is discussed in Section 6.

$$\begin{bmatrix} 0&1&1&1&1&1&1&1 \\ 0&1&2&2&2&2&2&2 \\ 0&1&2&2&2&2&2&2 \\ 1&1&2&2&2&2&3&3 \\ 1&1&2&2&2&2&3&4 \\ 1&1&2&2&2&2&3&4 \\ 1&2&2&2&2&3&3&4 \\ 1&2&3&3&3&3&3&4 \end{bmatrix}$$

Fig. 1. $x_{[1,2],[4,5]} \cong y_{[2,3],[7,8]}$

$$\begin{bmatrix} 0&1&1&1&1&1&1&1 \\ 0&1&2&2&2&2&2&2 \\ 0&1&2&2&2&2&2&2 \\ 1&1&2&2&2&2&2&2 \\ 1&1&2&2&2&2&2&3 \\ 1&1&2&2&2&2&3&3 \\ 1&2&2&2&2&3&3&3 \\ 1&2&3&3&3&3&3&3 \end{bmatrix}$$

Fig. 2. $x_{[1,2],[8,8]} \underset{3}{\cong} y_{[2,3],[8,8]}$

$$\begin{bmatrix} 0&1&1&1&1&1&1&1 \\ 0&1&3&3&3&3&3&3 \\ 0&1&3&3&4&4&4&4 \\ 4&4&4&5&5&5&7&7 \\ 4&4&4&5&5&5&7&9 \\ 4&4&4&8&8&8&9&9 \\ 4&5&5&8&8&8&9&9 \\ 4&5&7&8&8&9&9&10 \end{bmatrix}$$

Fig. 3. $x_{[4],[6],[8]} \cong y_{[1],[4],[7]}$

$$\begin{bmatrix} 0&1&1&1&1&1&1&1 \\ 0&1&3&3&3&3&3&3 \\ 0&1&3&3&3&3&3&3 \\ 4&4&4&4&4&4&4&4 \\ 4&4&4&4&4&4&5&7 \\ 4&4&4&6&6&6&7&7 \\ 4&5&5&6&6&6&7&7 \\ 4&5&7&7&7&7&7&7 \end{bmatrix}$$

Fig. 4. $x_{[1],[4,5]} \underset{3}{\cong} y_{[2],[7,8]}$

Definition 8. *For $\sigma, \tau \prec (n)$ and $1 \leq t \leq n-1$, let $\sigma^t \prec \sigma$ be defined by: $i \in \sigma^t \Leftrightarrow [i, i+t-1] \prec \sigma$. If $|\sigma| = |\tau|$, then for the unique $f : \sigma \to \tau$, we define σ^t_τ as: $i \in \sigma^t_\tau \Leftrightarrow [i, i+t-1] \prec \sigma$ and $[f(i), f(i)+t-1] \prec \tau$. We define $\tau^t_\sigma \prec \tau$ by $\tau^t_\sigma \equiv f(\sigma^t_\tau)$.*

Example 4. Let $\sigma, \tau \prec (16)$ be given in their block representations with $\sigma = [1,4],\ [7,10],\ [12,15]$ and $\tau = [2,8],\ [12,16]$. Then σ^2, τ^2, σ^2_τ and τ^2_σ are: $\sigma^2 = [1,3],\ [7,9],\ [12,14]$; $\tau^2 = [2,7], [12,15]$; $\sigma^2_\tau = [1,3], [7,8], [12,14]$; $\tau^2_\sigma = [2,4], [6,7], [13,15]$. Then σ^3, τ^3, σ^3_τ and τ^3_σ are: $\sigma^3 = [1,2], [7,8], [12,13]$; $\tau^3 = [2,6],\ [12,14]$; $\sigma^3_\tau = [1,2], [7,7], [12,13]$; $\tau^3_\sigma = [2,3], [6,6], [13,14]$. A careful inspection of σ^2_τ, τ^2_σ and σ^3_τ, τ^3_σ demonstrates the general result that $\sigma^t_\tau \underset{t}{\cong} \tau^t_\sigma$ [12].

For $x, y \in [q]^n$ with $x_\sigma = y_\tau$, we have that $i \in \sigma^t_\tau$ if and only if i is the first index in a common t-string , $x_{[i,i+t-1]} = y_{[f(i),f(i)+t-1]}$, of the common subsequence $x_\sigma = y_\tau$ of x and y. Thus $|\sigma^t_\tau|$ *is the number of common t-strings that occur in the common subsequence $x_\sigma = y_\tau$ of* x and y. In particular, $\left|\sigma^2_\tau\right|$ *is the number of common 2-strings that occur in the common subsequence $x_\sigma = y_\tau$ of* x and y. This measure is of interest to us because when two DNA strands have a secondary structure in a duplex, the thermodynamic weight (e.g., free energy) of *nearest neighbor stacked pairs* of that secondary structure is a measure (*not the measure*) of the thermodynamic stability of the duplex with the given secondary structure. Since every secondary structure in the DNA duplex $x : \overleftarrow{\overline{y}}$ between x and the complement, $\overline{y}$, of y corresponds to a common subsequence, $x_\sigma = y_\tau$, between x and y, we have that $\left|\sigma^2_\tau\right|$ gives us the number of nearest neighbor stacked pairs in the $x : \overleftarrow{\overline{y}}$ duplex with the secondary structure associated with $x_\sigma = y_\tau$. In general, $|\sigma^t_\tau|$ gives us the number of *common t-stems* in the $x : \overleftarrow{\overline{y}}$ duplex with the secondary structure associated with $x_\sigma = y_\tau$. See Section 6. We now show how we compute the "total weight" of the common 2-strings that occur in the common subsequence $x_\sigma = y_\tau$ of x and y.

Definition 9. *Suppose $2 \leq t \leq n-1$. Given a string $[a,b] \prec (n)$ and $x \in [q]^n$, let $d_q(x_{[a,b]})$ be the unique number in $[q^t]$ whose q-ary decimal representation is $x_a x_{a+1} x_{a+2} ... x_b$. For $x \in [q]^n$, let $x^{(t)} \in [q^t]^{n-t}$ be defined as $x^{(t)} \equiv \left(d_q(x_{[i,i+t-1]})\right)_{i=1}^{n-t+1}$. For example, if x =2, 3, 3, 0, 3, 0, 2, 2, 1, 1, 2, 0, 2$\in [4]^{13}$, then $x^{(2)}$ =11, 15, 12, 3, 12, 2, 10, 9, 5, 6, 8, 2$\in [16]^{12}$ and $x^{(3)}$ = 47, 60, 51, 12, 50, 10, 41, 37, 22, 24, 34$\in [64]^{11}$.*

Definition 10. *Suppose $2 \leq t \leq n-1$. Let Ω be a weight function on $[q^t]$. Then for $x, y \in [q]^n$, we define: $\psi^t_\Omega(x,y) \equiv \max\{\left\|x^{(t)}_{\sigma^t_\tau}\right\|_\Omega : x_\sigma = y_\tau\}$ and $\Psi^t_\Omega(x,y) \equiv \min\left(\left\|x^{(t)}\right\|_\Omega, \left\|y^{(t)}\right\|_\Omega\right) - \psi^t_\Omega(x,y)$. If $\|x_\sigma\|_\Omega = |\sigma|$,then we write $\psi^t(x,y)$ and $\Psi^t(x,y)$ for $\psi^t_\Omega(x,y)$ and $\Psi^t_\Omega(x,y)$ respectively.*

Proposition 4. *[12] Suppose $2 \leq t \leq n-1$. Let Ω be a weight function on $[q^t]$. Then for $x, y \in [q]^n$, we have:*

$$\psi^t_\Omega(x,y) = \phi^t_{\Omega,q^t}(x^{(t)}, y^{(t)}) \tag{4}$$

The application of Proposition 4 is that structural and thermodynamic aspects of DNA duplexes can be determined by means of Proposition 3. Consider the situation for $t = 2$. For a weight function, Ω, on $[q^2]$, when we think of a 2-string in $[q]^2$ as an element in $[q^2]$, we have that $\left\| x^{(2)}_{\sigma^2_\tau} \right\|_\Omega$ is the sum of the Ω weights of the common 2-strings $x_{[i,i+1]} = y_{[f(i),f(i)+1]}$ of $x_\sigma = y_\tau$. Therefore $\psi^t_\Omega(x, y)$ is maximum Ω weighted sum of the common 2-strings that can occur in any common subsequence of x and y. So Proposition 4 leads to two results: From Equation 4 in Proposition 4, we can, by virtue of Equation 3 in Proposition 3, calculate $\psi^t_\Omega(x, y)$. By application of Proposition 2, we have that $\Psi^t_\Omega(x, y)$ is a metric.

Definition 11. *A q-ary code of Ω weighted t-stem distance d is a subset C of $[q]^n$ such that: $x \neq y \in C \Longrightarrow \Psi^t_\Omega(x, y) \geq d$. If $t = 2$, we call such a C a q-ary code of Ω weighted stacked pair distance d. Thus if C is a Ω weighted code of t-stem distance d, then: $x \neq y \in C \Longrightarrow \left\| x^{(t)} \right\|_\Omega - \phi^t_{\Omega,q^t}(x^{(t)}, y^{(t)}) \geq d$ and $\left\| y^{(t)} \right\|_\Omega - \phi^t_{\Omega,q^t}(x^{(t)}, y^{(t)}) \geq d$. If $\| x_\sigma \|_\Omega = |x_\sigma|$, we simply call such a C a t-stem code of distance d.*

From a DNA duplex point of view, with $\Omega \equiv F$ being the thermodynamic weight of the (virtual) stacked pairs of nucleotides (see Table 1,) $\left\| x^{(2)} \right\|_\Omega$ is the absolute value of nearest neighbor free energy of the duplex $x : \overleftarrow{\overline{x}}$. Thus, if C is a (A,C,G,T) quaternary code of F weighted stacked pair distance d, then $x \neq y \in C$ implies that the thermodynamic stability of each of the duplexes $x : \overleftarrow{\overline{x}}$ and $y : \overleftarrow{\overline{y}}$ is at least "d greater than" the thermodynamic stability of the duplex $x : \overleftarrow{\overline{y}}$. Moreover, if C is a (A,C,G,T) quaternary t-stem code of distance d, then $x \neq y \in C$ implies $x : \overleftarrow{\overline{x}}$ and $y : \overleftarrow{\overline{y}}$ each have at least "d more" common t-stems than are in any secondary structure for duplex $x : \overleftarrow{\overline{y}}$. The main point of application is that $\psi^2_F(x, \overline{y})$ is a measure of the nearest neighbor stability of the DNA duplex $x : \overleftarrow{\overline{y}}$ and $\psi^t(x, y)$ is the maximum number of t-stems that can form in any secondary structure of $x : \overleftarrow{\overline{y}}$. For a 2-stem code C of distance $d = (n-1) - k$, we have for $x \neq y \in C$, that the *maximum number* of stacked pairs in a secondary structure of the duplex $x : \overleftarrow{\overline{y}}$ is at most k while the number of stacked pairs in each of the $x : \overleftarrow{\overline{x}}$ and $y : \overleftarrow{\overline{y}}$ duplex is $n - 1$. See Section 6.

Theorem 2. *[12] In $[q]^n$, there is a 2-stem code, C, of distance $d = (n-1) - k$ with:* $|C| \geq q^k \left(\sum_{j=1}^{k} q^{-j} \binom{k-1}{j-1} \binom{n-k}{j}^2 \right)^{-1}$.

6 Applications to DNA Hybridization Distance Modeling

Single strands of DNA are, abstractly, (A, C, G, T)-quaternary sequences, with the four letters denoting the respective nucleic acids. DNA sequences are oriented; e.g., $5'AACG3'$ is distinct from $5'GCAA3'$, but it is identical to $3'GCAA5'$.

The orientation of a DNA strand is usually indicated by the $5' \rightarrow 3', 3' \rightarrow 5'$ notation that reflects the asymmetric covalent linking between consecutive bases in the DNA strand backbone. In this paper, when we write DNA molecules without indicating the direction, it is assumed that the direction is $5' \rightarrow 3'$. Furthermore, DNA is naturally double stranded. That is, each sequence normally occurs with its reverse complement, with reversal denoting that two strands are oppositely directed, and with complementarity denoting that the allowed pairings of letters, opposing one another on the two strands, are $\{A, T\}$ or $\{G, C\}$—the canonical Watson-Crick *base pairings.* Therefore, to obtain the reverse complement of a strand of DNA, first reverse the order of the letters and then substitute each letter with its complement. For example, the reverse complement of $AACGTG$ is $CACGTT$. If $x = AACGTG$, then we let $\overline{x}$ denote it reverse complement $CACGTT$. We let $\overleftarrow{x}$ denote x listed in reverse $3' \rightarrow 5'$ order. *As DNA sequences, x and $\overleftarrow{x}$ are identical, i.e.,* $x = 5^{'}x_1, ..., x_n3^{'} \equiv \overleftarrow{x} = 3^{'}x_n, ..., x_1 5'$

A *Waston-Crick (WC) duplex* results from joining reverse complement sequences in opposite orientations, e.g., $x{:}\overleftarrow{\overline{x}} = \begin{matrix} 5'AACGTG3' \\ 3'TTGCAC5' \end{matrix}$. Whenever two, not necessarily complementary, oppositely directed DNA strands "mirror" one another sufficiently, they are capable of coalescing into a DNA duplex. The process of forming DNA duplexes from single strands is referred to as *DNA hybridization.* The greatest energy of duplex formation is obtained when the two sequences are reverse complements of one another and the DNA duplex formed is a WC duplex. However, there are many instances when the formation non-WC duplexes are energetically favorable. In this paper, a non-WC duplex is referred to as a *cross-hybridized (CH) duplex.*

An *$n-$DNA code* is a collection of single stranded DNA sequences of length n. In DNA hybridization assays, the general rule is that formation of WC duplexes is good, while the formation of CH duplexes is bad. A primary goal of DNA code design is be assured that a fixed temperature can be found that is well above the melting point of all CH and well below the melting point of all WC duplexes that can form from strands in the code. (It is also desirable for all WC duplexes to have melting points in a narrow range.) Thus the formation of any WC duplex must be significantly more energetically favorable then all possible CH duplexes. A DNA code with this property is said to have *high binding specificity.* High binding specificity is akin to a high signal-to-noise ratio. See [1], [2], [3], [4], [9], [10], and [21] for more comprehensive discussions of DNA codes.

A natural simplification for formulating binding specificity is to base it upon the maximum number of WC (inter-strand, non-covalent hydrogen) base pair bonds between complementary letter pairs which may be formed between two oppositely directed strands. Then for $x, y \in C$, an upper bound on this maximum number of base pair bonds that can form in the $x : \overleftarrow{y}$ duplex is the maximum length of a common subsequence to x and $\overline{y}$. In short, two single stranded DNA sequences x and y of length n can form d base pairs bonds in a duplex only if $\phi^1(x, \overline{y}) \leq n - d$. (See Definitions 5 and 6.) This doesn't mean that x and $\overleftarrow{y}$ will form d base pair bonds in a hybridization assay, it just says they could never form more than d base pair bonds.

If the binding specificity were solely dependent on the number of base pair bonds, then n-DNA codes constructed by using $\Phi^1(x, y)$ as the distance function could be used in hybridization assays with assured high binding specificity. This is because if $n-d$ is large enough, then one could find a temperature that exceeds the d base pair bonding threshold of all $x : \overleftarrow{y}$ CH duplexes, but is below the melting point of each $x : \overleftarrow{\overline{x}}$ WC duplex in which n base pair bonds form. Such a method was experimentally implemented in [5].

However, while the melting point of DNA duplexes depends, in part, on the number of base pair bonds, the state of the art model of DNA duplex thermodynamics is the Nearest Neighbor Model (NN). In the NN model, thermodynamic (e.g., free energy) values are assigned to *loops* rather than base pairs. We now briefly discuss some key aspects of the NN model. See [2], [23] and [25] for more comprehensive discussions about the NN model.

Consider two oppositely directed DNA strands $x = 5^{'}x_1, x_2, ..., x_i, ..., x_n 3'$ and $\overleftarrow{\overline{y}} = 3^{'}\overline{y_1}, \overline{y_2}, ..., \overline{y_j}, ..., \overline{y_n} 5^{'}$ where $\overline{y_j}$ denotes the complement to base y_j. A *secondary structure* of the DNA duplex $x : \overleftarrow{\overline{y}}$ is a sequence of pairs of *complementary* bases $((x_{i_r}, \overline{y_{j_r}}))$ where (x_{i_r}) and $(\overline{y_{j_r}})$ are subsequences of x and $\overleftarrow{\overline{y}}$ respectively. Clearly the duplex $x : \overleftarrow{\overline{y}}$ can have many secondary structures. An important issue is to understand *which* secondary structure in the most energetically favorable.

The collection of complementary pairs in a given secondary structure of a duplex partitions the duplex in to pairs of substrings (or subduplexes) that have the $(x_{i_r}), (\overline{y_{j_r}})$ and $(x_{i_{r+1}}), (\overline{y_{j_{r+1}}})$ as endpoints. For example, in the $x : \overleftarrow{\overline{y}}$ duplex presented as: $\begin{matrix} 5^{'}x_1, x_2, ..., x_{i_1}3^{'}*, ..., *5^{'}x_{i_r}, ..., x_{i_{r+1}}3^{'}*, ..., *5^{'}x_{i_k}, ..., x_n 3^{'} \\ 3^{'}\overline{y_1}, \overline{y_2}, ..., \overline{y_{j_1}}5^{'}*, ..., *3^{'}\overline{y_{j_r}}, ..., \overline{y_{j_{r+1}}}5^{'}*, ..., *3^{'}\overline{y_{j_k}}, ..., \overline{y_n}5^{'} \end{matrix}$

each pair $\begin{matrix} 5^{'}x_{i_r}, ..., x_{i_{r+1}}3' \\ 3^{'}\overline{y_{j_r}}, ..., \overline{y_{j_{r+1}}}5' \end{matrix}$ of substrings (separated by *) is an elementary substructure called a *loop* of the given secondary structure $((x_{i_r}, \overline{y_{j_r}}))$ of the given duplex $x : \overleftarrow{\overline{y}}$. If each of the strings in a loop are of length 2, e.g., $\begin{matrix} 5^{'}x_{i_r}, x_{i_r+1}3^{'} \\ 3^{'}\overline{y_{j_r}}, \overline{y_{j_r+1}}5' \end{matrix}$, then that loop is called a *stacked pair.*

Example 5. We use mix of lower case and upper case letters to help identify the secondary structure. Consider the duplex $\begin{matrix} 5'ggCaTaTcatACt3' \\ 3'TccAAttGgtaGa5' \end{matrix}$ where the secondary structure is $(g, c), (g, c), (a, t), (a, t), (c, g), (a, t), (t, a), (t, a)$. Loops are $E_1, ..., E_8$ and are listed left to right: $\begin{matrix} g*gg\ *gGc\ *aTa*aTc*ca*at*tACt \\ Tc*cc\ *cAAt*tt\ \ *tGg*gt*ta*aGa \end{matrix}$ $= E_1 \overset{*}{\underset{*}{}} E_2 \overset{*}{\underset{*}{}} E_3 \overset{*}{\underset{*}{}} E_4 \overset{*}{\underset{*}{}} E_5 \overset{*}{\underset{*}{}} E_6 \overset{*}{\underset{*}{}} E_7 \overset{*}{\underset{*}{}} E_8$. The free energy, ΔG, of the duplex predicted by the NN model is approximately $\sum_{i=1}^{8} \Delta G_i$ where ΔG_i is the free energy assigned to loop E_i. However, in many cases, the most stabilizing features of the structure come from the stacked pairs i.e., E_2, E_6, and E_7, and the free energies of stacked pairs are the most accurately measured. See

[23]. The free energies for most non-stacked loops are approximated from the free energy for stacked pairs with the same terminal pairs. See [25] For example, the free energy of $E_3 = \frac{5'gCa3'}{3'cAAt5'}$ would be approximated by adding a "penalty" to the free energy for the measured free energy for the stacked pair $\frac{5'ga3'}{3'ct5'}$ (that does not appear in the above secondary structure.) In most cases, the penalty takes on a positive value while all of the free energies for stacked pairs are negative. It is therefore reasonable to assume that if one only considers the free energies for the stacked pairs, then their sum would be a lower bound for the NN free energy for the given duplex with the given secondary structure.

Consider two identically directed DNA strands $x = 5^{'}x_1, x_2, ..., x_i, ..., x_n3'$ and $y = 5^{'}y_1, y_2, ..., y_j, ..., y_n3^{'}$. For computational purposes, we define the idea of a *virtual secondary structure* between these two identically directed strands even thought no such structure would naturally form. A virtual secondary structure of the *virtual DNA duplex* $x : y$ is a sequence of pairs of identical bases $((x_{i_r}, y_{j_r}))$ where (x_{i_r}) and (y_{j_r}) are subsequences of x and y respectively. In other words, a virtual secondary structure of the virtual duplex $x : y$ *is a common subsequence* $x_\sigma = y_\tau$ *of* x *and* y. Then the virtual duplex $x : y$ has the virtual secondary structure $((x_{i_r}, y_{j_r}))$ if and only if the actual duplex $x : \overleftarrow{\overline{y}}$ (where $x = 5^{'}x_1, x_2, ..., x_i, ..., x_n3'$ and $\overleftarrow{\overline{y}} = 3^{'}\overline{y_1}, \overline{y_2}, ..., \overline{y_j}, ..., \overline{y_n}5^{'}$) has the actual *secondary structure* of pairs of complementary bases $((x_{i_r}, \overline{y_{j_r}}))$ where (x_{i_r}) and $(\overline{y_{j_r}})$ are subsequences of x and $\overleftarrow{\overline{y}}$ respectively. A stacked pair, $\frac{5^{'}x_{i_r}, x_{i_r+1}3^{'}}{3'\overline{y_{j_r}}, \overline{y_{j_r+1}}5'}$, exists in the actual secondary structure $((x_{i_r}, \overline{y_{j_r}}))$ if and only if the corresponding *virtual stacked pair,* $\frac{5^{'}x_{i_r}, x_{i_r+1}3^{'}}{5'y_{j_r}, y_{j_r+1}3'}$, exists in the virtual secondary structure of the virtual duplex $x : y$. Thus, there exists a virtual stacked pair in a virtual secondary structure $x_\sigma = y_\tau$ if and only if $(x_i, x_{i+1}) = (y_{f(i)}, y_{f(i)+1})$ is a common 2-string of the common subsequence $x_\sigma = y_\tau$ where $f : \sigma \to \tau$ is unique. Now the connection to Section 5 is clear.

In [23], the free energies (all negative) of natural stacked pairs are given. Identifying virtual stacked pairs with their natural representation, the *virtual free energy* ($\digamma$) values can be associated to the negative of their corresponding values (ΔG) for actual stack pairs. The actual values are given in KCAL/mole

Table 1. Thermodynamic weight of virtual stacked pairs

$\digamma$	A	C	G	T
A	1.00	1.44	1.28	0.88
C	1.45	1.84	2.17	1.28
G	1.30	2.24	1.84	1.44
T	0.58	1.30	1.45	1.00

measured at 37C and with specified ionic concentrations. In Table 1, give the values (from [23]) with their corresponding virtual stacked pairs. Since the virtual stacked pair is a pair of identical 2-strings $(x_i, x_{i+1}) = (y_{f(i)}, y_{f(i)+1})$, we can represent this virtual stacked pair by (x_i, x_{i+1}) and denote its virtual free energy by $\digamma(x_i, x_{i+1})$. The $(i,j)^{th}$ entry of Table 1 is the value of $\digamma(i,j)$, e.g., $\digamma(C,T) = 1.28$. (So $\digamma(C,T)$ denotes the free energy associated with the $\begin{matrix} 5'CT3' \\ 3'GA5' \end{matrix}$ naturally occurring stacked pair. We take $\digamma$ as our weight function on $[4^2]$.

Let $x : \overleftarrow{\overline{y}}$ be an actual duplex and let $\Delta G(x : \overleftarrow{\overline{y}})$ be the NN computation of the free energy of the $x : \overleftarrow{\overline{y}}$ duplex. The main point of all of this is that it is quite reasonable to assume that in most cases: $-\psi^2_\digamma(x,y) \leq \Delta G(x : \overleftarrow{\overline{y}})$. From a DNA duplex point of view, with $\digamma$ being the thermodynamic weight of the virtual stacked pairs of nucleotides, we have: $\left\|x^{(2)}\right\|_\digamma = -\Delta G(x : \overleftarrow{\overline{x}})$. Thus if C is a $\digamma$ weighted stacked paired (A, C, G, T) quaternary code of distance d, then: $x \neq y \in C \Longrightarrow \Psi^2_\digamma(x,y) \geq d$. This implies that the thermodynamic stability, $-\Delta G(x : \overleftarrow{\overline{x}})$ and $-\Delta G(y : \overleftarrow{\overline{y}})$, of each (all) of the WC duplexes $x : \overleftarrow{\overline{x}}$ and $y : \overleftarrow{\overline{y}}$, respectively would each be at least "d greater than" the thermodynamic stability ,$-\Delta G(x : \overleftarrow{\overline{y}})$, of the (any) $x : \overleftarrow{\overline{y}}$ CH duplex where $x \neq y \in C$. Thus, n-DNA codes closed under complementation ($x \in C \Leftrightarrow \overline{x} \in C$) constructed by using $\Psi^2_\digamma(x,y)$ as the distance function could be used in hybridization assays with high binding specificity.

Example 6. Given DNA sequences $x = GTTATAGGCCGAG$ and $y = CGTC$ $GTGTATATT$ of length 13, consider the virtual secondary structure $x_\sigma = y_\tau$ with $\sigma = [1,6]$ and $\tau = [5,6],[8,11]$. We have that $\sigma^2_\tau = [1,1],[3,5]$ and $\tau^2_\sigma = [5,5],[8,10]$. We use lower case letters to exhibit the common subsequences that represent the virtual secondary structures represented by $x_\sigma = y_\tau$: $\begin{matrix} gttataGGCCGAG \\ CGTCgtGtataTT \end{matrix}$. Identify $0 \equiv A$, $1 \equiv C$, $2 \equiv G$ and $3 \equiv T$ and convert the DNA sequences accordingly. Then $x^{(2)} = \mathbf{11}, 15, \mathbf{12}, \mathbf{3}, \mathbf{12}, 2, 10, 9, 5, 6, 8, 2$ and $y^{(2)} = 6, 11, 13, 6, \mathbf{11}, 14, 11, \mathbf{12}, \mathbf{3}, \mathbf{12}, 3, 15$ where the (bold faced) block isomorphic subsequence, $x^{(2)}_{\sigma^2_\tau} \cong y^{(2)}_{\tau^2_\sigma}$,represents the four virtual stacked pairs gt, ta, at, ta in the displayed virtual secondary structure $x_\sigma = gttata = y_\tau$. Using the $\digamma$ in Table 1, we have that $\left\|x^{(2)}_{\sigma^2_\tau}\right\|_\digamma = 3.48$. However, $\psi^2_\digamma(x,y) = 3.61$. This is because the virtual secondary structure $x_\alpha = y_\beta$ with $\alpha = [1,2], [10,11], [13,13]$ and $\beta = [2,5], [7,7]$ depicted using lower case letters as: $\begin{matrix} gtTATAGGCcgAg \\ CgtcgTgTATATT \end{matrix}$ has $\alpha^2_\beta = [1,1]$, $[10,10]$ and $\beta^2_\alpha = [2,2]$, $[4,4]$. Then $x^{(2)}_{\alpha^2_\beta} = 11, 6 = y^{(2)}_{\beta^2_\alpha}$ represents the virtual stacked pairs gt and cg in the virtual secondary structure $x_\alpha = gtcgg = y_\beta$. Finally, we have that $\psi^2_\Omega(x,y) = \left\|x^{(2)}_{\alpha^2_\beta}\right\|_\Omega = 3.61$.

Example 7. Given DNA sequences $x = AATCCAACATTATTGC$ and $y = GTCACATCATCAAGCC$ and using the $\digamma$ in Table 1, we have $\left\|x^{(2)}\right\|_\digamma =$

18.39, $\left\|y^{(2)}\right\|_{F} = 20.7$ and $\psi_F^2(x,y) = 8.19$. Thus $\Psi_F^2(x,y) = 10.20$. We also have that $x^{(2)}$ =0,3,13,5,4,0,1,4,3,15,12,3,15,14,9; $y^{(2)} = 11, 13, 4, 1, 4, 3, 13, 4, 3, 13, 4, 0, 2, 9, 5$. There are at most six stacked pairs in any virtual secondary structure between x and y, i.e., $\psi^2(x,y) = \phi^2(x^{(2)}, y^{(2)}) = 6$. A virtual secondary structure that has six stacked pairs is $x_\sigma = x_{[3,4],[7,10],[12,13],[15,16]} = y_{[2,6],[8,9],[14,15]} = y_\tau$. These six stacked pairs are represented by the common block isomorphic subsequence $x^{(2)}_{\sigma^2_\tau} = x^{(2)}_{[3,3],[7,9],[12,12],[15,15]} \cong y^{(2)}_{[2,2],[4,6],[8,8],[14,14]} = y^{(2)}_{\tau^2_\sigma}$. In this case, $\psi_F^2(x,y) = \phi_F^2(x^{(2)}, y^{(2)}) = \left\|x^{(2)}_{[3,3],[7,9],[12,12],[15,15]}\right\|_F = 8.19$. We also have that $x^{(3)} = 2, 9, 36, 20, 16, 1, 4, 18, 10, 39, 33, 10, 42, 45$ and $y^{(3)} = 57, 35, 17, 4, 18, 9, 35, 18, 9,\ 35, 16, 3, 13, 53$. Since $\psi^3(x,y) = \phi^3(x^{(3)}, y^{(3)}) = 2$, we have that most number of 3-stems in any secondary virtual secondary structure between x and y is 2. Note that$x^{(3)}_{[7,8]} \underset{3}{\cong} y^{(3)}_{[4,5]}$. Note that the virtual secondary structure $x_\sigma = x_{[3,4],[7,10],[12,13],[15,16]} = y_{[2,6],[8,9],[14,15]} = y_\tau$ has exactly two 3-stems, namely $x_{[7,9]} = y_{[4,6]} = ACA$ and $x_{[8,10]} = y_{[5,7]} = CAT$.

7 t-Stem DNA Code Generation Software

We briefly describe a program which we make freely available. (Contact the corresponding author for instruction on how to access this program or go to: http://cluster.ds.geneseo.edu:8080/ParallelDNA/) The program(s) generates DNA codes. Some of the inputs are:

1. Length of DNA codewords: n; 2. Stem sizes checked: $t_1, t_2, ...$; 3. Corresponding thresholds for each stem size: $s_1, s_2, ...$; 4. Maximum CH free energy parameter: ΔG_{CH}; 5. Nearest neighbor WC free energy lower bound parameter: $\underline{\Delta G_{wc}}$ 6. Nearest neighbor WC free energy upper bound parameter: $\overline{\Delta G_{wc}}$.

What is generated is a DNA code C such that:

1. $x \in C \Rightarrow |x| = n$ and $\overline{x} \in C$. Thus the WC complement of each strand in the code is also in the code.

2. $x \neq y \in C \Rightarrow \psi^{t_i}(x,y) \leq s_i$. Thus the maximum number of $t_i - stems$ in each CH duplex from C is at most s_i.

3. $x \neq y \in C \Rightarrow \psi_F^2(x,y) \leq \Delta G_{CH}$. Thus each CH duplex in C has a free energy of formation above $-\Delta G_{CH}$

4. $x \in C \Rightarrow \underline{\Delta G_{wc}} \leq \left\|x^{(2)}\right\|_F \leq \overline{\Delta G_{wc}}$. Thus each WC duplex in C has a free energy of formation between $\overline{-\Delta G_{wc}}$ and $\underline{-\Delta G_{wc}}$.

Example 8. Below is a DNA code generated by one of our programs with the inputs: n=16; t_1,t_2,t_3=1,2,3; s_1,s_2,s_3=10,6,2; ΔG_{CH}=8; $\underline{\Delta G_{wc}} = 18$; $\overline{\Delta G_{wc}} = 22$. No codeword contains GGG or CCC as a substring. The complement of any strand is either to the immediate right or left of the given strand. There are 30 codewords in the code below.

GGCCAAAAAAAAAAAA, TTTTTTTTTTTTGGCC, GGCAAAGGTTTTCCAA, TTGGAAAACCTTTGCC, CATTTTAAGGAACCGG, CCGGTTCCTTAAAATG, TCCTCTTTCTTTACCA, TGGTAAAGAAAGAGGA, TAGAATCCGTCAATTT, AAATTGACGGATTCTA, GGTTACGGTGGTGTTT, AAACACCACCGTAACC, TTTGTCACTTGTGGAG, CTCCACAAGTGACAAA, AGTATTTCGATCTTCC, GGAAGATCGAAATACT, CAGGCGTTGATGAACA, TGTTCATCAACGCCTG, TAACTATGTAGCATGG, CCATGCTACATAGTTA, CAACAATAGGAGGCTT, AAGCCTCCTATTGTTG, GGACTTAGGCAGACGT, ACGTCTGCCTAAGTCC, GAGCGAGGTAGATTAG, CTAATCTACCTCGCTC, GATACACACGGCATAT, ATATGCCGTGTGTATC, CGAGTGGCTCTCTCAT, ATGAGAGAGCCACTCG,

References

[1] M. Andronescu, A. Condon and H. Hoos, RNAsoft, submitted to NAR for the web-based software special issue, available at http://www.rnasoft.ca/

[2] M. Andronescu, Algorithms for predicting the secondary structure of pairs and combinatorial sets of nucleic acid strands, Masters Thesis, University of British Columbia, (2003.)

[3] E. Baum. DNA sequences useful for computation, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 44, 235-242, (1999.)

[4] A. Brenneman and A. Condon, Strand Design for biomolecular computation, Theoretical Computer Science, 287, 39-58, (2002).

[5] H. Cai, et al., Flow Cytometry-Based Minisequencing: A New Platform for High Throughput Single Nucleotide Polymorphism Scoring, Genomics, 66, 135-143, (2000.)

[6] A. D'yachkov and D. Torney, On Similarity Codes, IEEE Trans. on Information Theory 46, 1558-1564, (2000.)

[7] R. Deaton, et al., A PCR Based Protocol for in Vitro Selection of Noncrosshybridizing Oligonucleotides, DNA Computing, DNA 8, M. Hagiya, A. Ohuchi (eds.), LNCS 2568, Springer, Berlin 196-204 (2002.)

[8] R. Deaton, et al., A Software Tool for Generating Noncrosshybridizing Libraries of DNA Oligonucleotides, DNA Computing, DNA 8, M. Hagiya, A. Ohuchi (eds.), LNCS 2568, Springer, Berlin 252-261 (2002.)

[9] A. D'yachkov, et al., On a Class of Codes for Insertion-Deletion Metric, 2002 IEEE Intl. Symp. Info. Th., Lausanne, Switzerland, (2002.)

[10] A. D'yachkov, et al., Exordium for DNA Codes, Journal of Combinatorial Optimization, 7, no.4, 369-380 (2003.)

[11] A. D'yachkov, et al., Reverse-Complement Similarity Codes, IEEE Trans.on Information Theory to appear

[12] A. D'yachkov, et al., An Insertion-Deletion Like Metric, manuscript

[13] P. Erdos, D. Torney, and P. Sziklai, A Finite Word Poset, Elec. J. of Combinatorics, 8, (2001.)

[14] M. Garzon, et al., A new metric for DNA computing, in Genetic Programming 1997: Proceedings of the Second Annual Conference, pp. 479-490, AAAI, 1997. Stanford University, July 13-16, 1997.

[15] D. Gusfield, Algorithms on Strings, Trees, and Sequences, Cambridge, (1997.)

[16] Hartemink, A., Gifford, D., A thermodynamic simulation of deoxyoligonucleotide hybridization for DNA computation, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 48, 25-37 (1999.)
[17] H. Hollman, A relation between Levenshtein-type distances and insertion and deletion correcting capabilities of codes, IEEE Trans. on Information Theory, 39 1424-1427, (1993.)
[18] V. Levenshtein, Efficient reconstruction of sequences from their subsequences or supersequences, Journal of Combinatorial Theory, Series A, 93, 310-332 (2001.)
[19] V. Levenshtein, Binary Codes Capable of Correcting Deletions, Insertions, and Reversals, Soviet Phys.–Doklady, 10 707-710, (1966).
[20] V. Levenshtein, Bounds for Deletion-Insertion Correcting Codes, 2002 IEEE Intl. Symp. Info. Th., Lausanne, Switzerland, (2002).
[21] A. Macula, DNA-TAT Codes, USAF Technical Report, TR-2003-57, AFRL-IF-RS http://stinet.dtic.mil/cgi-bin/fulcrum_main.pl (2003.)
[22] A. Macula, et al., DNA Code Gen., available at https ://community.biospice.org
[23] J. SantaLucia Jr., A unified view of polymer, dumbbell, and oligonucleotide DNA nearest-neighbor thermodynamics, Proc. Natl. Acad. Sci. USA, Vol. 95, pp 1460-1465 (1998.)
[24] M. Waterman, Introduction to Computational Biology, Chapman-Hall, London, (1995.)
[25] A. Zuker, B. Mathews and C. Turner, Algorithms and Thermodynamics for RNA Secondary Structure Prediction: a Practical Guide

DNA Extraction by XPCR

Giuditta Franco[1], Cinzia Giagulli[2], Carlo Laudanna[2], and Vincenzo Manca[1]

[1] Department of Computer Science, University of Verona
franco@sci.univr.it, vincenzo.manca@univr.it
[2] Section of General Pathology,
Department of Pathology, University of Verona
{cinzia.giagulli, carlo.laudanna}@univr.it

Abstract. The extraction of DNA strands including a given sequence of bases is a crucial step in the Adleman-Lipton extract model of DNA computing. In this paper, a special type of PCR is presented with a related algorithm which performs a specified extraction from a given pool of DNA double stranded (shortly dsDNA) molecules. This kind of PCR, called Cross Pairing PCR (shortly XPCR) was tested in several situations, and used in a biotechnological procedure which implements the extraction algorithm.

1 Introduction

Since the introduction of the Adleman-Lipton extract model [1, 8], the fundamental schema of a DNA algorithm, for solving an instance of a combinatorial problem, is the following: i) Generation of a pool DNA strands encoding all possible solutions (the solution space), ii) Extraction of those that are the true solutions of the given instance. This second step is performed by a sequence of elementary extraction sub-steps, where at each sub-step all the strands where a specific sub-strand occurs are selected from the pool and constitute the input for the next extraction sub-step. These two steps are usually of complexity that is linear in time with respect to the size of the given instance. This is the conceptual strength of DNA computing, because the pools that are elaborated in the steps of the procedure are of a size that is exponential with respect to the size of the given instance. Generation of the solution space can be performed in several manners, by using the power of DNA recombination [1, 10], or by a sequence of steps according a Mix-and-Split procedure [3]. Extraction remains the critical point of this paradigm. For example, the method which uses the biotin-avidine affinity to select some strands by means of the complementary substrands bond to beads has efficiency $88 \pm 3\%$. Moreover, in the context of the Adleman-Lipton extract model, as it is reported in [7], if we call p the percentage of sound extractions (extracting each of the required DNA strands is equally likely) and we assume that for each distinct string s in a test tube there are 10^l ($l = 13$ proposed by Adleman in [1]) copies of s, then no matter how large l is and no matter how close to 1 p is, there always exists a class of 3-SAT problems such that DNA computing error must occur.

In this paper we address the following particular problem. Given a specified sequence γ of bases, and an input pool P of different dsDNA molecules with a same length n and sharing a common prefix and suffix, we want to produce an output pool P' where only the strands which include the given sequence γ are represented. We will show that by using PCR in a particular manner, combined with gel-electrophoresis, we are able to solve this problem.

PCR is one of the most important and efficient tool in biotechnological manipulation and analysis of DNA molecules. The main ingredients of this reaction are polymerase enzymes which implement a very simple and efficient duplication algorithm on double oriented strings. The PCR procedure is based on: i) templates, ii) a copy rule applied to templates, iii) initial short strings (primers) that say where the copying process has to start. Polymerase enzyme 'writes' step by step, in the $5'-3'$ direction, by coping (in complementary form) the bases of the template which drives the copy process. The bilinearity of DNA molecules and the antiparallel orientation of their two linear components are essential aspects of the logic underlying the whole process [5, 11].

The idea of using PCR as a "very elegant and effective detection method" to check the existence of a solution was considered in [12], where a theoretic method 'blocking' the wrong sequences with PNA strands is proposed, and a paper on experimental aspects of DNA computing by blocking was announced.

In the next section we propose the XPCR procedure, that is a variant of standard PCR where two dsDNA molecules and two primers are used in such a way that one primer hybridizes with one single strand of a DNA molecule, and the other primer with one single strand of the other DNA molecule. A similar idea, but in a very different context, was considered in [9]. In section 3 we present an extraction algorithm based on XPCR, and we give some experimental results that show its validity.

2 Cross Pairing PCR

Firstly we introduce some terminology and notation.

As usual, we intend that upper strands are in the direction from left to the right ($5'-3'$) and lower strands are in the opposite direction. We use the terms 'string' and 'sequence' in an almost equivalent manner, however, we speak of *strings* when we want to stress their abstract concatenation structure as words of a free monoid over a finite alphabet (the Watson Crick alphabet in our specific case). We adopt Greek letters for strings, and $\bar{\alpha}$ will indicate the reversed complementary string of α. *Strands* are physical (oriented and rotational, single or double) DNA instances of strings. We say γ-strand any strand (single or not) which is an instance of string γ, and γ-superstrand any strand which includes a γ-strand. Moreover, shortly we say that $\alpha \in P$ when α-strands are present in the pool P.

In the following we will explain in an informal manner the steps of the extraction algorithm we propose in this paper, by using as much as possible a pictorial language, but it is interesting to note that by extending the bilinear notation introduced in [5, 11] we could express all the process we consider in a complete formal way.

If one puts in a test tube many copies of $\alpha \ldots \gamma$-strands and $\gamma \ldots \beta$-strands (that finish and start respectively with a same substring γ), and two primers α and $\bar{\beta}$, then PCR performs the process one can see in figure 1, where $\alpha \ldots \gamma \ldots \beta$-

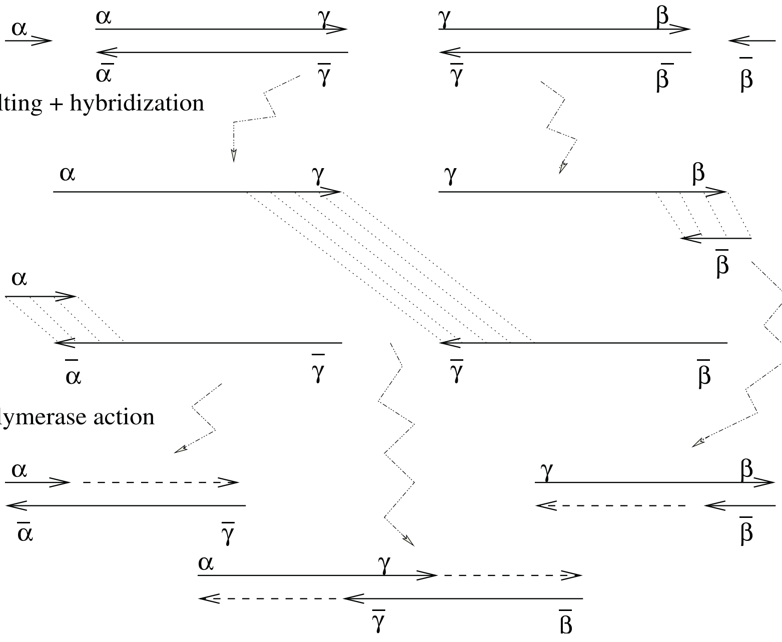


Fig. 1. Basic step of XPCR

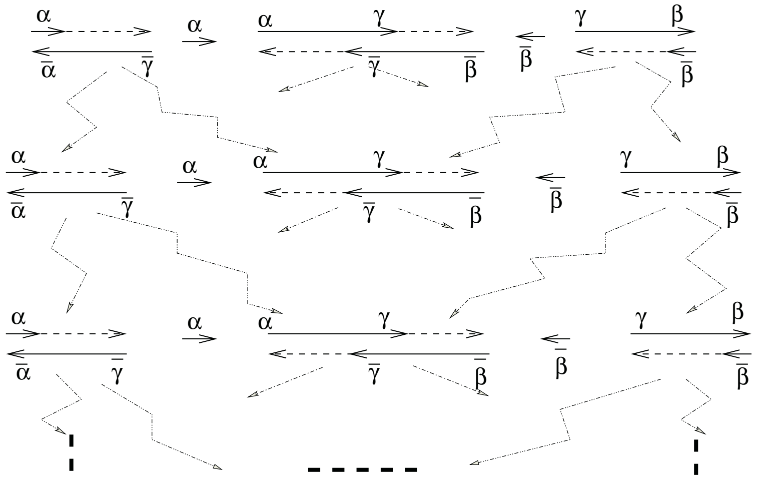


Fig. 2. Amplification of XPCR. Long strands in the middle are seeds of exponential amplifications

strands are firstly generated, by a sort of overlapping juxtaposition of the two initial strands, and then amplified. This step has been verified in laboratory in different situations and variants, with three sizes of γ (229, 95, 15 bp, see footnotes 1, 4, 6 in section 3.1).

As one can see in figure 1, differently from standard PCR the primers hybridize with single strands of two different dsDNA molecules, so liberating the respective partners in each molecule. At this point, these single strands can hybridize each other by means of their (reversed) complementary parts γ and $\bar{\gamma}$, and the polymerase uses the single strand components of this structure as templates in order to complete the double string. This process is the key point of the extraction algorithm of the next section (see step 4). Note that only the long strings are amplified in each step (see figure 2); in fact, one of the strands of the initial short pieces in every step is the template for the generation of another short piece, and the two generated short pieces will join to form a long string including γ.

3 Extraction Algorithm

Let us start with a pool P which is constituted by dsDNA molecules having same length n, α-strands at beginning and β-strands at end. Given a string γ, let us assume that P is γ-*invariant*, that is, either γ does not occur at the same position in different strands of P, or if it is not the case, then $\alpha\tau_1\gamma\tau_2\beta, \alpha\tau_3\gamma\tau_4\beta \in P$ implies that $\alpha\tau_1\gamma\tau_4\beta, \alpha\tau_3\gamma\tau_2\beta \in P$. The hypothesis of a common prefix and suffix and the γ-*invariance* of the pool are not restrictive assumptions in the context of DNA computing. The following procedure gives as output a pool P' where all the γ-superstrands of P are represented.

In the pictures related to each step of the algorithm we do not mention the products given by secondary linear amplification, because ignoring them will not affect the validity of the procedure. We use the notation PCR(α, β) to indicate a PCR performed with α as forward primer and β as reverse primer.

1. **PCR($\alpha, \bar{\gamma}$)**
   After this step we find in the test tube an exponential amplification of the dsDNA $\alpha \ldots \gamma$ (see figure 3) that are shorter than the initial molecules (products linearly amplified keep the initial length).
2. **PCR ($\gamma, \bar{\beta}$)**
   After this step we find in the test tube an exponential amplification of the dsDNA $\gamma \ldots \beta$ (see figure 4) that are shorter than the initial molecules (products linearly amplified keep the initial length).
3. **Gel-electrophoresis for selecting the short strands of lengths l_1, l_2 such that $l_1 + l_2 - l = n$ where l is the length of γ**
   In this step only the strings $\alpha \ldots \gamma$ and $\gamma \ldots \beta$ are selected.
4. **XPCR($\alpha, \bar{\beta}$)**
   In this step all the γ-superstrands of P, that are the longest ones in the current pool (see figure 5), are exponentially amplified.
5. **Gel-electrophoresis for selecting the n-long strands**.

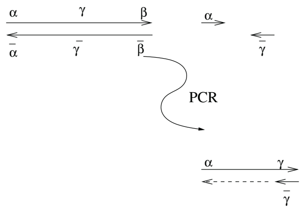


Fig. 3. First step based on PCR

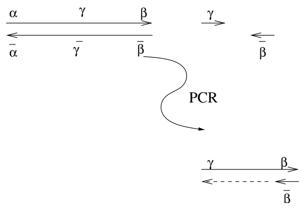


Fig. 4. Second step based on PCR

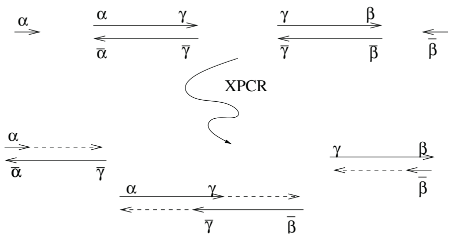


Fig. 5. Fourth step based on XPCR

3.1 Encoding and Experiments

A very important aspect of our method is the type of encoding we need in order to avoid unexpected phenomena of annealing. In fact, primers have to work in a very specific manner. We adopted a comma free encoding, following some of the general principles given in [2, 4], by using a program that checked the strings of the pool and primers according to the following strategy. A test $T(n, m)$ is positive when a situation is found such that in a window of n consecutive positions there are at least m discordance positions This means that when $T(n, m+1)$ is negative then a window there exists with more than $n - m + 1$ concordance positions (in this case the program localizes and shows all these windows). Several tests were performed with different values of the parameters n and m related to the size of our primers.

An important *caveat* is the 'primer rotation' phenomenon. When it occurs, a forward primer can play the role of another reverse primer or *vice-versa.* We

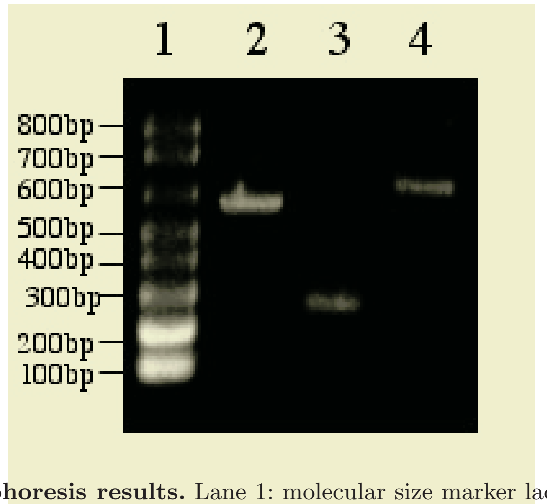


Fig. 6. Electrophoresis results. Lane 1: molecular size marker ladder (100b). Lane 2: $\alpha\phi\gamma\psi$-strands of human RhoA (582bp), lane 3: $\gamma\psi\beta$-strands (253bp), lane 4: cross pairing amplification of $\alpha\phi\gamma\psi\beta$-strands (606bp): 606 = 582 +253 - 229

collected the outcomes of many experimental trials that suggested us the values of parameters that ensure a reliable behavior of the primers. However, oligos and primers we used in the experiment do not have common strings of 5 bases long (apart the expected annealing part, where the concordance is total in a window with the same length of the primer).

In order to test the validity of XPCR, a first experiment was carried on with $\alpha\phi\gamma\psi$-strands, $\gamma\psi\beta$-strands, and primers α and $\bar{\beta}$ (data are shown in figure 6)[1], where $\alpha\phi\gamma\psi$ was RhoA human gene which regulates many essential cellular processes and controls cell activation in response to environmental cues. In the following $-n$ position of a sequence indicates the position n in the backward direction.

Other experiments were performed with pools of sequences $\alpha\tau_2\ldots\tau_9\beta$ (150 long) generated by a combination of strands $X_2, X_3, \ldots, X_9$, Y_3, Y_4, …, Y_8, Z_2, Z_4, Z_6, Z_7, Z_9 [2] in such a way that τ_i is equal to X_i or Y_i or Z_i (for each i we have at least two choices). In particular, we started from a pool of eight different

[1] α = ATGGCTGCCATCCGGAAG, γ = GAAGGATCTTCGGAATGATG … at position -229 of Rho A, and $\bar{\beta}$ = GAACAGAAACTTATCTCAGAGGAA.

[2] X_2 = CAAGATATGG, X_3= TCGTCTGCTAGCATG, X_4 = TCACGCCACGGAACG, X_5 = GTGAGCGCGAGTGTG, X_6 = ATATGCAATGATCTG, X_7 = ATCCGTCCCGATAAG, X_8 = CAAGTCAGATTGACC, X_9 = GCACGTAACT, Y_3 = CCCGATTAGTACAGC, Y_4 = TACTGATAAGTTCCG, Y_5 = TCGCTCCGACACCTA, Y_6 = TCAGCCGGCTTGCAC, Y_7 = AACTGATACGACTCG, Y_8 = TATTGTCACGCATCG, Z_2 = CAAGAGATGG, Z_4 = TCACGCCACGGAACT, Z_6 = TTAGCCGGCTTGCAC, Z_7 = TACTGATACGACTCG, Z_9 = GTACGTAACT, α = GCAGTCGAAGCTGTTGATGC, β = AGACGCTGCCGTAGTCGACG.

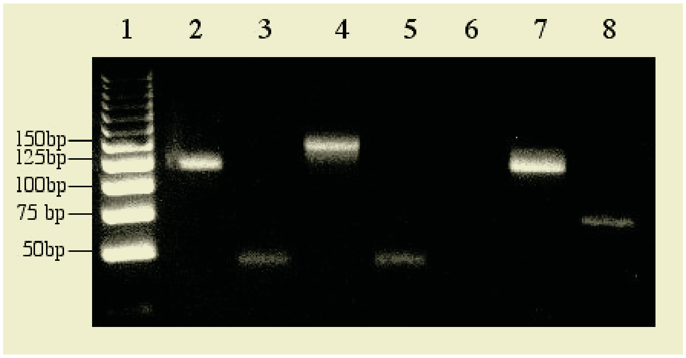


Fig. 7. Electrophoresis results. Lane 1: molecular size marker ladder (25 bp). Lane 2: amplification of $\alpha \ldots \gamma$ strands (120 bp); lane 3: amplification of $\gamma \ldots \beta$ strands (45 bp); lane 4: cross pairing amplification of $\alpha \ldots \gamma$ and $\gamma \ldots \beta$ (150 bp). Lane 5: positive control by PCR$(\gamma, \bar{\beta})$, with γ at position -45; lane 6: negative control by PCR$(\gamma', \bar{\beta})$; lane 7, 8: positive controls by PCR$(\gamma_1, \bar{\beta})$ and PCR$(\gamma_2, \bar{\beta})$ respectively, with γ_1 at position -125 and γ_2 at position -75

types of strands[3] where γ-superstrands $\alpha \ldots \gamma$ and $\gamma \ldots \beta\delta$ were obtained by standard PCRs and again also in this case the expected results: $\alpha \ldots \gamma \ldots \beta\delta$-strands were produced by XPCR [4].

Finally, the complete algorithm was tested on a pool[5] in which γ' is present only in all the sequences that are not γ-superstrands[6] and all the γ-superstrands are either γ_1-superstrands or γ_2-superstrands[7]; all the steps were proved to be correctly performed (see lanes 2, 3, 4, 5 of figure 7), in fact all and only γ-superstrands were extracted from the pool. In order to verify the correctness of our results we performed three PCRs on the final test tube. The first one with primers γ' and $\bar{\beta}$ showed that only γ-superstrands were present in the final pool, because there was no amplification of γ'-superstrands (see lane 6 of figure 7). The last two PCRs with primers γ_1, $\bar{\beta}$, and γ_2, $\bar{\beta}$, respectively, showed that all the initial γ-superstrands were present in the final tube (see lanes 7, 8 of figure 7).

[3] $\alpha\ Z_2\ X_3\ X_4\ X_5\ X_6\ X_7\ Y_8\ Z_9\ \beta$, $\alpha\ X_2\ Y_3\ X_4\ X_5\ Z_6\ Y_7\ Y_8\ Z_9\ \beta$, $\alpha\ X_2\ Y_3\ X_4\ X_5\ X_6\ Z_7\ X_8\ X_9\ \beta$, $\alpha\ Z_2\ X_3\ Y_4\ Y_5\ Y_6\ Y_7\ X_8\ X_9\ \beta$, $\alpha\ Z_2\ X_3\ Z_4\ Y_5\ Y_6\ X_7\ Y_8\ Z_9\ \beta$, $\alpha\ X_2\ Y_3\ Y_4\ Y_5\ Y_6\ X_7\ X_8\ X_9\ \beta$, $\alpha\ Z_2\ X_3\ Y_4\ Y_5\ Y_6\ X_7\ Y_8\ Z_9\ \beta$, $\alpha\ X_2\ Y_3\ Y_4\ Y_5\ Y_6\ Y_7\ X_8\ X_9\ \beta$.

[4] $\gamma =$ GAACGGTGAGCGCGAGTGTG ... in position -95 in all the strands where it occurs, $\delta =$ CTTGTCTTTGAATAGAGTCTCCTT.

[5] $\alpha\ Z_2\ X_3\ X_4\ X_5\ X_6\ X_7\ Y_8\ Z_9\ \beta$, $\alpha\ X_2\ Y_3\ X_4\ X_5\ Z_6\ Y_7\ Y_8\ Z_9\ \beta$, $\alpha\ Z_2\ X_3\ Y_4\ Y_5\ Y_6\ Y_7\ X_8\ X_9\ \beta$, $\alpha\ X_2\ Y_3\ Y_4\ Y_5\ Y_6\ X_7\ X_8\ X_9\ \beta$, $\alpha\ X_2\ Y_3\ Y_4\ Y_5\ Y_6\ Y_7\ X_8\ X_9\ \beta$.

[6] $\gamma = Y_8$, $\gamma' = Y_4$.

[7] $\gamma_1 =$ GATGGTCGTCTGCTAGCATG , $\gamma_2 =$ TTAGCCGGCTTGCAAACTG.

4 Conclusions

In general and abstract terms XPCR is a method for performing, in a cheap and efficient manner, the following transformation on strings that is essentially the splicing combinatorial mechanism in the sense of the original formulation introduced by Tom Head in [6] (more precisely, a case of *null context splicing* rule):

$$\alpha\,\phi\,\gamma\,\psi\,\beta,\quad \alpha\,\delta\,\gamma\,\eta\,\beta \qquad\longrightarrow\qquad \alpha\,\phi\,\gamma\,\eta\,\beta,\quad \alpha\,\delta\,\gamma\,\psi\,\beta.$$

We showed that this method is useful for selecting γ-superstrands from a given pool, but we think that XPCR could have also a more general relevance in the context of DNA computing. of solution spaces of In principle, consecutive DNA extraction from a given pool, by means of XPCR, should work correctly, but problems could arise if the encoding is not robust enough for avoiding unexpected annealing. Therefore, we intend to develop, in the next future, encoding methods that make iterated XPCR reliable. Applications and extensions of cross pairing amplification and of the extraction algorithm based on it will also be topics of future researches.

5 Appendix (Experimental Protocols)

Reagents. 25 bp and 1 kb marker DNA ladder (Promega); agarose (Gibco brl); PCR buffer, $MgCl_2$ and dNTP (Roche); Taq DNA Polymerase (produced in laboratory); all the synthetic DNA oligonucleotides 150 bases long and all the primers were from Primm s.r.l. (Milano, Italy).

Annealing of Synthetic DNA Oligonucleotides. Two complementary synthetic 150 bases long DNA oligonucleotides ($5' - 3'$ and $3' - 5'$) were incubated at 1:1 molar ratio at 90° for 4 min in presence of 2.5 mM of $MgCl_2$ and then at 70° for 10 min. The annealed oligos were slowly cooled to 37°, then further cooled to 4° until needed.

PCR Amplification. PCR amplification was performed on a PE Applied Biosystems GeneAmp PCR System 9700 (Perkin Elmer, Foster City, CA) in a 50μl final reaction volume containing 1.25U of Taq DNA Polymerase, 1.5 mM $MgCl_2$, 200 μM each dNTP, PCR buffer, 50 ng DNA template, 0.5-1 μM of forward and reverse primers. The reaction mixture was preheated to 95° for 5 min. (initial denaturing), termocycled 25 times: 95° for 1 min. (denaturing), 58° for 1 min. (annealing), 72° for 15 sec (elongation); final extension was performed at 72° for 10 min.

Preparation and Running of Gels. Gels were prepared in 7x7 cm or 6x10 cm plastic gel cassettes with appropriate combs for well formation. Approximately 20 or 35 ml of 4% agarose solutions were poured into the cassettes and allowed to polymerize for 10 min. Agarose gels were put in the electrophoresis chamber and electrophoresis was carried out at 10 volt/cm^2, then the bands of the gels were detected by a gel scanner. The DNA bands (PCR products) of interest were excised from the gel and the DNA was purified from the gel slices by Promega Kit (Wizard SV Gel and PCR Clean-Up System).

Acknowledgments

We kindly thank Prof. Angelo Spena of the Biotechnological Department of the University of Verona for his interesting suggestions on PCR phenomenon, and Luca Bianco of the Department of Computer Science of the University of Verona for his collaboration in the encoding of sequences.

References

1. Adleman L. M., *Molecular Computation of solutions to combinatorial problems*, Science **266**, pp 1021-1024, 1994.
2. Arita M., *Writing Information into DNA*, in: Aspects of Molecular Computing, N. Jonoska, G. Păun, G. Rozenberg (eds.), LNCS 2950, pp. 23-35, 2004.
3. R.S. Braich, N. Chelyapov, C. Johnson, P.W.K. Rothemund, L. Adleman, *Solution of a 20-Variable 3-SAT Problem on a DNA Computer*, Science **296**, pp 417-604, 2002.
4. R.S. Braich, C. Johnson, P.W.K. Rothemund, D. Hwang, N. Chelyapov, L. Adleman, *Solution of a Satisfiability Problem on a Gel-Based DNA Computer*, A. Condon, G. Rozenberg (eds): DNA Computing, 6th International Workshop on DNA-Based Computers, Leiden, Netherlands. Lecture Notes in Computer Science, Springer-Verlag **2054**, pp 27-42, 2000.
5. G. Franco, V. Manca, *An algorithmic analysis of DNA structure*, Soft Computing, to appear, 2004.
6. T. Head, *Formal language theory and DNA: An analysis of the generative capacity of specific recombinant behaviors*, Bulletin of Mathematical Biology **49**, pp 737-759, 1987.
7. D. Li, *Is DNA computing viable for 3-SAT problems?*, TCS **290**, pp 2095-2107, 2003.
8. R.J. Lipton, *DNA solutions of hard computational problems*, Science **268**, pp 542-544, 1995.
9. S. Liu, D. S. Thaler, A. Libchaber, *Signal and noise in bridging PCR*, BMC Biotechnology, **2**:13, 2002.
10. V. Manca, C. Zandron, *A Clause String DNA Algorithm for SAT*, N. Jonoska, N.C. Seeman (eds.): DNA Computing, 7th International Workshop on DNA-Based Computers, Tampa, Florida. Lecture Notes in Computer Science **2340**, Springer, pp 172-181, 2001.
11. V. Manca, *On the logic of DNA bilinearity*, Preliminary Proceedings, Masami Hagiya, Azuma Ohuchi (Eds.): DNA Computing, 8th International Meeting on DNA Based Computers, Hokkaido, Japan, p 330, 2002.
12. G. Rozenberg, H. Spaink, *DNA computing by blocking*, TCS **292**, pp 653-665, 2003.

A Method of Error Suppression for Self-assembling DNA Tiles

Kenichi Fujibayashi and Satoshi Murata

Interdisciplinary Graduate School of Science and Engineering,
Tokyo Institute of Technology,
Yokohama, 226-8502, Japan
fuji@mrt.dis.titech.ac.jp, murata@dis.titech.ac.jp

Abstract. Self-assembling DNA tile is an important method of molecular computation. In this method, DNA tiles self-assemble a large two-dimensional lattice by the specific hybridization between complementary strands. Hence hybridization is not a deterministic, but stochastic, process which depends on tile concentration and temperature. For that reason, it is difficult to obtain an aggregate with no error. Growth speed of the aggregate must be very low to obtain an aggregate that contains less errors. Winfree et al. proposed the Proofreading Tile Model (PTM), which achieves both a low error rate and fast growth speed by splitting tiles into pieces to enhance tile specificity. However, it remains difficult to implement because it requires a large set of completely orthogonal strands.

This paper presents a novel method called Layered Tile Model (LTM) to realize the reliable self-assembly of DNA tiles. We introduce layered tiles which are covered by other tiles called protective tiles. Those protective tiles verify the correctness of connections of the former tiles. Simulation and analysis are used to evaluate LTM performance. Results demonstrate that LTM offers similar performance compared to 2×2 PTM. It also has unique properties that are considered to be practical for implementation.

1 Introduction

Recently, Winfree et al. proposed a method of computation [1] based on DNA tile self-assembly. DNA tiles self-assemble into a two-dimensional lattice in a process that is designed to realize some computation. It is demonstrably equivalent to the computation process of the universal Turing machine. Various methods using DNA tile self-assembly have been proposed [2, 3, 4, 5, 6, 7, 8]. Those methods are typically based on correctness of bindings among tiles. However, that assumption is not valid because thermal fluctuation renders the binding process unreliable. Winfree proposed a model called kinetic Tile Assembly Model (kTAM) [9] to address that issue. Using kTAM, and thereby controlling physical conditions such as temperature and the monomer tile concentration, it was possible to produce an aggregate that was almost free from binding error. However, growth speed of the aggregate is necessarily sacrificed to maintain a low error rate. Winfree et al. proposed the the Proofreading Tile Model (PTM) to eliminate that disadvantage [10]. This model realizes both a low error rate and high-speed growth by splitting tiles into pieces, thereby improving logical selectivity among DNA tiles. It requires many orthogonal sticky ends, which complicates the design.

This paper proposes a method of error suppression, called Layered Tile Model (LTM), for self-assembly of DNA tiles. It controls erroneous connections among tiles by introducing an additional layer of tiles which cover those exposed sticky ends. This mechanism not only reduces erroneous connections, but also suppresses random aggregation. This study designs a molecular structure of DNA tiles for LTM, simulates its self-assembly process, and evaluates the consequent relationship between its growth speed and error rate.

2 Tile Assembly Model

Winfree proposed the abstract Tile Assembly Model (aTAM) and the kinetic Tile Assembly Model (kTAM) to describe a computation process using DNA tiles [9]. The aTAM is the simplest two-dimensional self-assembly model that is based on Wang's Tiling [11, 12]. The aTAM comprises a finite set of square tiles: each edge of these tiles has a specific label and strength $b_{\mathcal{D}}$ (Fig. 1a). The total strength b of each tile is given as the sum of matched labels $b_{\mathcal{D}}$. An aggregate grows from a seed tile by gathering monomer tiles on its boundary. A monomer tile is capable of binding at the growth front of the aggregate only if its binding strength b is greater than the value of a threshold τ. This model allows definition of a certain tile set which can form a uniquely defined structure. Hereafter, we specifically address such a tile set which produces an infinite Sierpinski fractal pattern. The set contains four rule tiles and three boundary tiles (Fig. 1b). At $\tau = 2$, starting from a corner tile (seed tile), we obtain an errorless aggregation (Fig. 1c).

The aTAM is extended to kinetic Assembly Model (kTAM: we refer to this model as Original Tile Model (OTM) in this paper) to consider thermodynamical factors (e.g. temperature, tile concentrations). In kTAM, tiles correspond to DNA double-crossover (DX) molecules [13]. As illustrated in Fig. 2a, the DX molecule is a planar molecule composed of two DNA double helices connected side by side. Four single DNA strands extrude from the tile; they are called *sticky ends*. Two sticky ends which have Watson-Crick complementary oligonucleotides can bind to each other by DNA hybridization. Binding strengths depend on the number of hydrogen bonds in hybridization; consequently, they depend on the length of the sticky end. That length is equivalent to $b_{\mathcal{D}}$ in aTAM (sticky end of length s has bonding strength $b_{\mathcal{D}} = 1$ [1]).

[1] In Winfree's models (OTM, PTM), $s = 5$; in our model (LTM), $s = 14$.

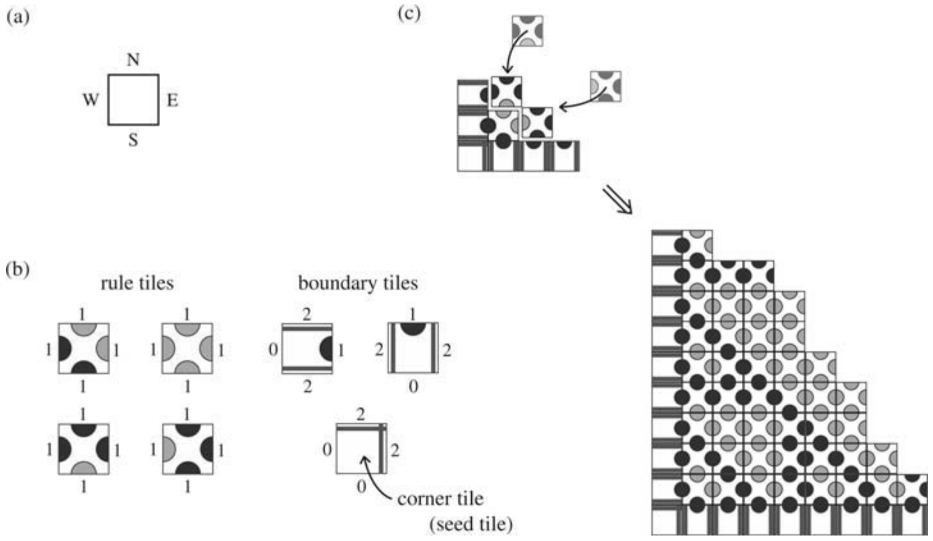


Fig. 1. (a) Abstract tile. Edges are labeled N (*North*), E (*East*), S (*South*), and W (*West*). (b) Tile set for Sierpinski pattern. (c) Growth of Sierpinski pattern from the corner tile by aTAM at $\tau = 2$. Rule tiles can bind only at concave steps of the boundary

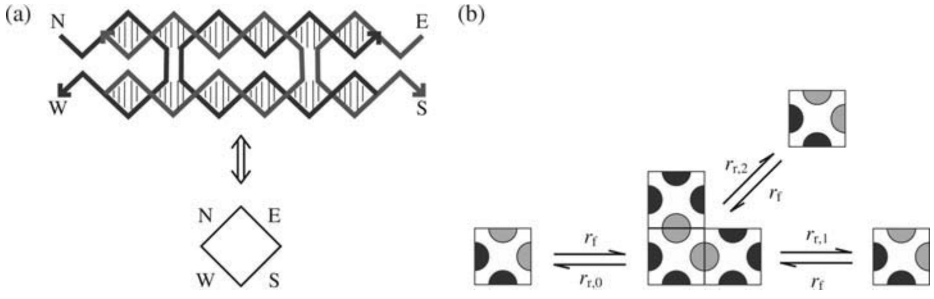


Fig. 2. (a) DX molecule and its representation as an abstract tile. Four sticky ends of DX molecule correspond to four edges of the abstract tile. (b) Rates for tile addition and tile dissociation in kTAM: r_f is the forward rate for association of any tile at any site, and $r_{r,b}$ is the reverse rate for dissociation of a tile which has total bonding strength of b

In kTAM, Winfree made several assumptions for simplification [9]: (i) Monomer (single tile) concentrations $[monomer]$, measured in M, are always constant; all monomer types are held at the same concentration. (ii) Interactions (addition and dissociation) occur only between an aggregate and monomer tiles. Interaction among aggregates is neglected. (iii) Forward rate constants k_f for all monomers are identical. In particular, the forward rate constants for correct and incorrect additions are equal. (iv) Reverse rate constants $k_{\mathrm{r},b}$ depend exponentially on b, where only matched edges are counted as b.

Under these assumptions, the rate of addition is given as

$$r_{\mathrm{f}} = k_{\mathrm{f}}[monomer] = k_{\mathrm{f}} e^{-G_{\mathrm{mc}}} \quad , \tag{1}$$

measured in /sec, where $G_{\mathrm{mc}} > 0$ is the entropic cost of putting a monomer at the binding site. It depends on monomer concentration ($[monomer] = e^{-G_{\mathrm{mc}}}$). The rate of dissociation $r_{\mathrm{r},b}$ /sec is given as

$$r_{\mathrm{r},b} = k_{\mathrm{r},b} = k_{\mathrm{f}} e^{-bG_{\mathrm{se}}} \quad , \tag{2}$$

which is also measured in /sec, where $G_{\mathrm{se}} > 0$ is the free energy to break one sticky end ($b_{\mathcal{D}} = 1$). G_{se} depends on temperature T and sticky end length s ($G_{\mathrm{se}} = (\frac{4000}{T} - 11)s$, approximately). Rates of addition and dissociation for various tiles are illustrated in Fig. 2b.

In kTAM, the ratio $\frac{G_{\mathrm{mc}}}{G_{\mathrm{se}}}$ is equivalent to the threshold τ in aTAM. When G_{se} is large and G_{mc} is small, tiles have a high tendency to associate and a low tendency to dissociate. In region $0 < \tau < 1$, aggregates grow quickly, but engender great error. In contrast, when G_{se} is small and G_{mc} is large, tiles have a low tendency to associate and a high tendency to dissociate. Therefore, aggregates do not grow in $2 < \tau$. Optimal growth is realized when τ is close to 2, implying that we must sacrifice growth speed to maintain a low error rate [2]. Winfree estimated the relation between the error rate ϵ and growth speed r_{g} as $r_{\mathrm{g}} \approx \beta\epsilon^2$ ($\beta = 1 \times 10^5$ /M/sec) [10].

Winfree proposed a second model, called the Proofreading Tile Model (PTM), to improve both the error rate and growth speed [10]. In PTM, these are realized by splitting each DNA tile into $n \times n$ pieces. In the case of 2×2 PTM, the relation becomes $r_{\mathrm{g}} \approx \beta\epsilon$, implying that much faster growth is possible at the same error rate. However, this method requires numerous independent sticky ends, which complicate the design.

3 Layered Tile Model

We propose the Layered Tile Model (LTM), which improves growth speed and the error rate. In this model, two different types of tiles (upper tiles and lower tiles) are superposed: the upper tiles control hybridization of sticky ends of the lower tiles. Sticky ends of upper tiles protect sticky ends of the lower tiles so that the addition of subsequently added tiles is allowed only if the lower tile is connected to the aggregate correctly. This mechanism allows suppression of deadlocks[3], which are engendered by mismatched tiles.

[2] At $\tau = 2$, reactions are at a local equilibrium; the aggregate cannot grow. $\tau = 2 - \epsilon$ ($0 < \epsilon < 1$) is required for correct self-assembly.

[3] A deadlock is a situation in which a mismatched tile is buried by other tiles before it dissociates from the aggregate.

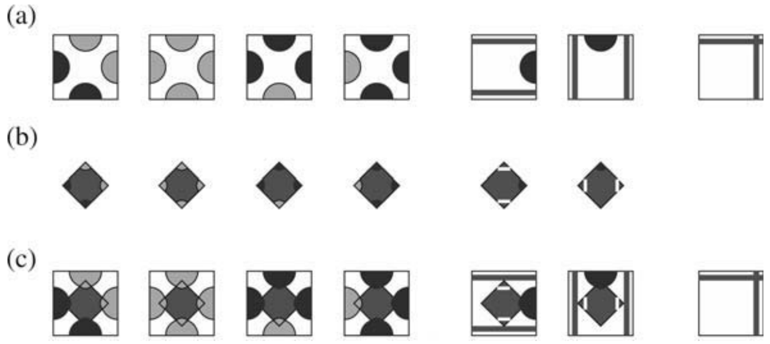


Fig. 3. (a) The original (lower) tiles. (b) The protective (upper) tiles. (c) The layered tiles

3.1 A Growth Process of the LTM

LTM uses 13 tile types (Fig. 3). Seven of them are the same tiles used in OTM (composing the lower layer); six of them are *protective tiles*, which correspond to each of them except for the corner tile (composing upper layer). Here, we consider a set of original tiles (a) to form the Sierpinski fractal pattern. They are protective tiles (b) used to protect the original tiles, and a pair of a protective tiles superposed on an original tile, called a *layered tile* (c).

We assume that all tiles, except for the corner tile, are layered at the initial condition. One of two states – p (protected) or u (unprotected) – is defined on each edge of the original tiles. p is the state in which the edge of the original tile is protected by the protective tile. u is the state in which the edge of the original tile is not protected. When a monomer tile is superposed by the protective tile, all edges have state p (Fig. 4a).

We apply the following rules:

Rule 1. If the edge S or W of a layered tile matches with other tiles or aggregates, then the state of the matched edge is switched from p to u.
Rule 2. Edges N and E of the layered tile are always protected (p); they allow no additional bonding.
Rule 3. When both S and W of a layered tile match ($p \rightarrow u$) then the protective tile dissociates; N and E of the original tile become u.

Fig. 4 illustrates how these rules work.

3.2 Implementation by DNA Tiles

We designed molecular structures shown in Fig. 5a to implement these rules. Squares (Black) and Diamonds (red) represent the original tile and protective tile, respectively. The protective tile actually has an *identical* molecular structure to the original tile. Lines attached to squares and diamonds represent sticky

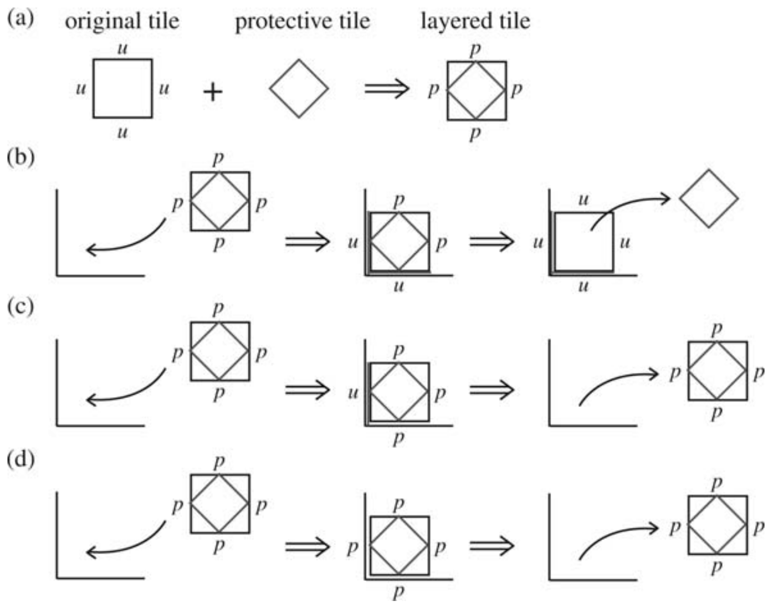


Fig. 4. (a) Monomer layered tile. (b) Two-edge matched: states at S and W are switched to u by **Rule 1**; the states at N and E are switched to u by **Rule 3**. Subsequently, the next connection on this tile is permitted. (c) One edge matched: the matched edge becomes u by **Rule 1**, but **Rule 3** is not applied. In this case, the layered tile cannot adhere to the aggregate by a single match. Moreover, burying of the layered tile never occurs because both N and E are p by **Rule 2**. (d) No edge matched: the layered tile cannot remain because all the edges have state p

ends; integers are bond strengths of each sticky end (see Appendix A about their determination). For rule tiles, the total bonding strength between original tiles and protective tiles is defined such that it does not become greater than $\tau = 2$.

Rule 1 $\sim$ **3** can be implemented as the DNA tiles. The following addresses only rule tiles (boundary tiles are not discussed here).

Implementation of Rule 1. The switching state at S or W of the original tile from p to u is realized by removing a sticky end of the protective tile from the original tile. This process is realized through *branch migration*[4] [14, 15]. Let the sticky end of the original tile at the growth front of the aggregate be S_{a},

[4] Branch migration is a process of selective replacement of DNA strands. It occurs whenever DNA strands have complementary sequences to others. Those strands compete with each other to achieve longer binding length; the longest matching pair is allowed to remain. This process is irreversible once it is completed.

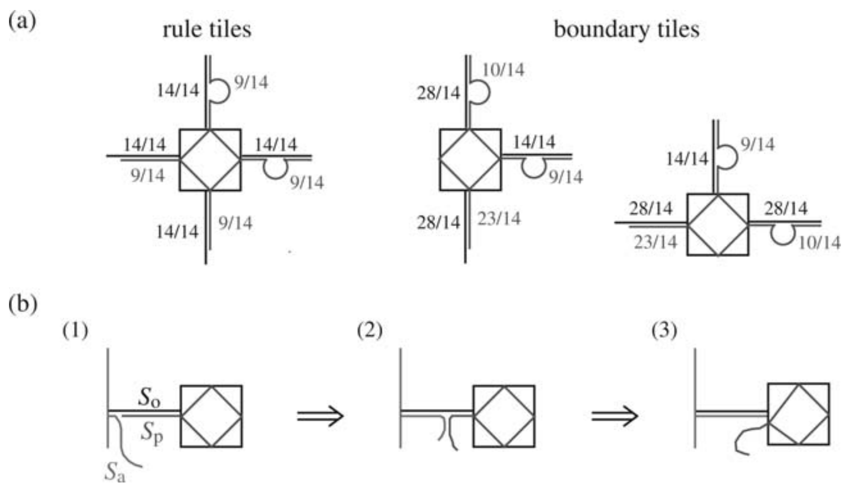


Fig. 5. (a) Schemata of layered tiles. The black square and red diamond represent the original tile and the protective tile, respectively. Integers beside sticky ends represent their bonding strengths. (b) A sticky end of the protective tile S_p is replaced by a correct sticky end of the aggregate S_a by branch migration

and the corresponding sticky end at W of the original tile be S_o. S_p denotes a sticky end of protective tile which also corresponds to S_a (Fig. 5b). S_p serves to connect the protective tile to the original tile, whereas S_p is 5-base shorter than S_o, which provides a sticky end for S_a. By branch migration, S_a replaces S_p and releases the protective tile at W (or S).

Implementation of Rule 2. Sticky ends at N and E of the protective tile protect sticky ends of the original tile. Binding lengths at both N and E must not be full length. Thereby, they release the protective tile only when sticky ends at both S and W are correctly engaged. We think the tips of sticky ends of the original tile should be covered to prevent unnecessary connections. A *loop* structure (Fig. 5a) may be effective[5] [16] to prevent hybridization with other sticky ends.

Implementation of Rule 3. Suppose that states at both S and W of the original tiles are already switched from p to u. The sum of bonding strengths at N and E is designed to be insufficient to hold the protective tile. Therefore, the protective tile dissociates and protection at N and E disappears. If only one matched end remains at S or W, the protective tile has sufficient bonding strength and remains on the spot (Fig. 4c).

[5] To form a double helix, single-strand DNA must twist around the a small loop.

4 Simulations

We have built a simulation system to evaluate LTM. Using the simulator, we can estimate the error rate and aggregate size. For comparison, we also calculate them for OTM using the same simulator.

For LTM, the following rules and assumptions are added to Winfree's OTM algorithm (Appendix B):

- In the case of *on-events*, six *layered* tiles and one corner tile are equally likely to be chosen.
- We introduce states p and u on the edges of original tiles and apply **Rule 1** $\sim$ **3** in Section 3.1.
- The duration for branch migration is neglected[6].
- Once protective tiles are dissociated from original tiles, they cannot associate again.

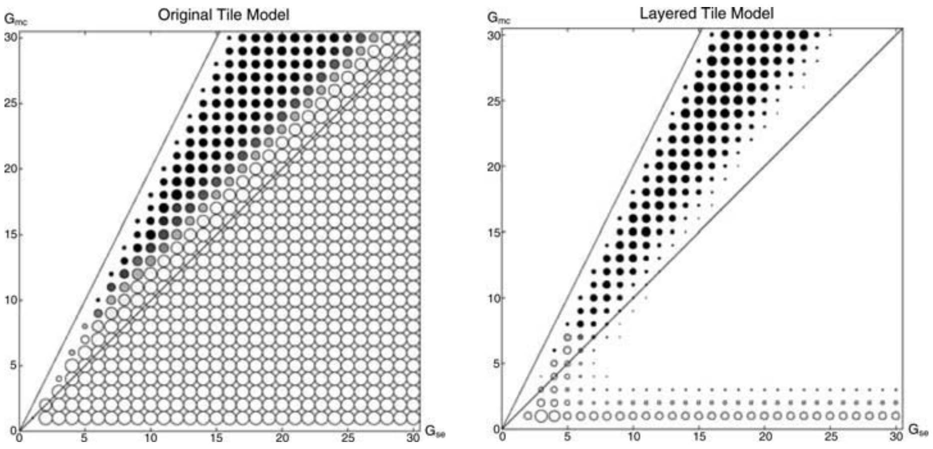


Fig. 6. Phase plot of OTM and LTM computed by simulations. Each disc represents the result of a single simulation; the disc size represents the size of the aggregate; the shading tone represents the error rate (Solid black indicates a zero error rate.)

Simulations were performed for both OTM and LTM to map out parameter spaces of G_{se} and G_{mc}. Fig. 6 shows error rates and aggregate sizes for $1 \leq G_{\text{se}}$, $G_{\text{mc}} \leq 30$. Each disc represents a result of single simulation up to the 20000th *on-event*. The disc size represents the size of the aggregate; the shading tone represents the error rate[7]. Solid black indicates a zero error rate. Lines with slopes 1 and 2 represent $\tau = \frac{G_{\text{mc}}}{G_{\text{se}}} = 1$ and 2.

[6] This assumption will be reconsidered in Section 5.

[7] The error rate is given as $error\,rate = \frac{mismatches}{tiles}$, where *mismatches* is a number of mismatched tiles, and *tiles* is the number of tiles in the aggregate. A mismatched tile is one which has four neighboring tiles, some of which have wrong edges.

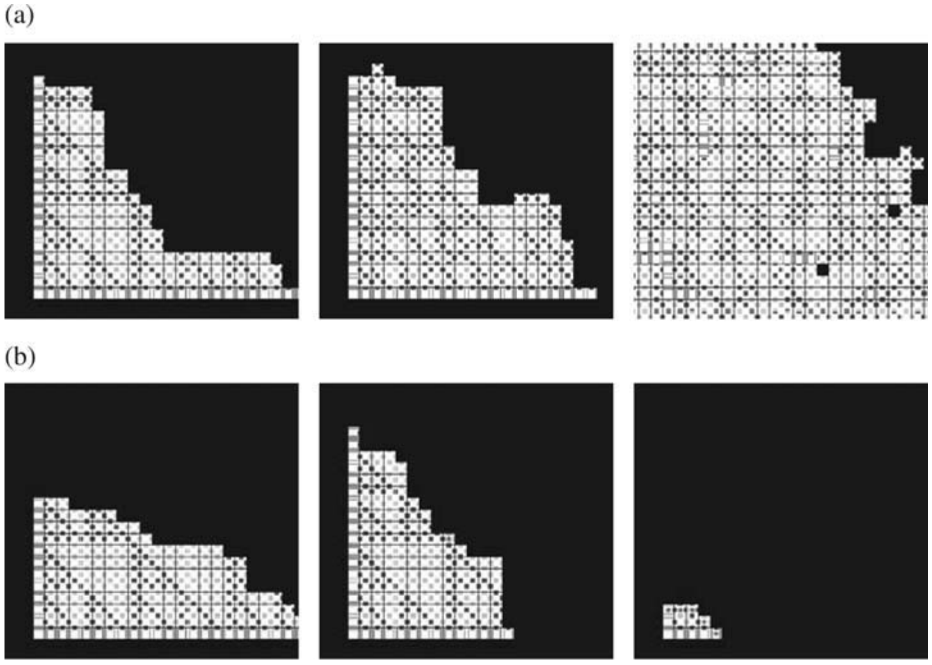


Fig. 7. Simulation results of OTM (a) and LTM (b) for different τ. From left to right, $\tau = \frac{G_{\mathrm{mc}}}{G_{\mathrm{se}}} = \frac{24}{17} = 1.412$, $\frac{19}{17} = 1.118$ and $\frac{14}{17} = 0.824$. In the bottom right frame, protective tiles (red diamonds) cannot dissociate from original tiles; growth has stalled

As shown in this figure, LTM properties differ greatly from those of OTM. In the $2 < \tau$ region, neither model produces an aggregate. In $1 < \tau < 2$, aggregates can grow with low error rates in both models. The region near the line $\tau = 1$, LTM shows similar performance to that of OTM. The most distinguished difference between them is in $0 < \tau < 1$. In this region, OTM produces a random aggregate; whereas, in LTM, we observe almost no growth. This is because protective tiles cannot be dissociated from original tiles. For instance, even if two edges of a layered tile match, total bonding strength between protective tiles and original tile at N and E is $\frac{18}{14}$, which means no association at $\tau < \frac{18}{14}$. Whereas, in $1 \leq G_{\mathrm{mc}} \leq 3$, random growth is observed in LTM because tile concentrations are sufficiently high to produce random aggregation.

Fig. 7 shows snapshots from simulation for different values of τ. At $\tau = 1.412$, both models show error-free aggregates. At $\tau = 1.118$, LTM has a better error rate than OTM. OTM produces a random aggregate at $\tau = 0.824$. However, in LTM, the aggregate does not grow at all. This is regarded as *suppressed growth* realized by protective tiles. To dissociate a protective tile, the total strength of a tile must not exceed τ. For small τ, tiles almost cease assembly because protective tiles are always attached to original tiles.

5 Analysis

We evaluate LTM analytically using the *kinetic trapping model* [9]. This model allows calculation of quantitative properties such as the error rate and growth speed.

We consider the associate/dissociate process at a single growth site, which is called a concave *step*, on the boundary. Initially, a transition diagram that describes transitions among possible states at a step is required. We define ten states for the analysis. Four of them are most important: (E) the step is empty; (C) a tile at the step is correct, namely both S and W are matched; (A) a tile at the step is partly correct, namely S or W is mismatched; and (I) a tile at the step is incorrect, namely both S and W are wrong. We also require states LC_1, LC_2, LA_1 and LA_2, where 'L' denotes *layered*, i.e., a protective tile is superposed on an original tile. LC_1 and LA_1 denote states before branch migration (Fig. 5b-1); they become LC_2 and LA_2 after branch migration, respectively (Fig. 5b-3). We also add two states FC and FI where 'F' means *frozen.* In frozen states, a tile is buried by other tiles at the next event; thereby, it cannot dissociate.

Now, we have to evaluate transition rates between above states (see Fig. 8):

- $\mathrm{E} \rightarrow \mathrm{LC}_1$. The rate is r_f because only one kind of layered tile is correct in the tile set of Sierpinski pattern. Similarly, the rate for $\mathrm{E} \rightarrow \mathrm{LA}_1$ is $2r_\mathrm{f}$ (two layered tiles) and $\mathrm{E} \rightarrow \mathrm{I}$ is $4r_\mathrm{f}$.
- $\mathrm{LC}_1 \rightarrow \mathrm{LC}_2$ and $\mathrm{LA}_1 \rightarrow \mathrm{LA}_2$ are branch migration, which is *irreversible* once completed. The transition rate of branch migration is given as $\frac{1}{t_\mathrm{bm}}$ /sec, where $t_\mathrm{bm} = 0.001$ sec is a rough estimation of duration necessary for branch migration[8].
- For $\mathrm{LA}_2 \rightarrow \mathrm{E}$ rate is $r_{\mathrm{r},1}$, because a layered tile can be dissociated from the aggregate after branch migration.
- For $\mathrm{LC}_2 \rightarrow \mathrm{C}$ and $\mathrm{LA}_2 \rightarrow \mathrm{A}$, a protective tile dissociates from an original tile. Considering the bonding strength necessary to remove the protective tile from the original tile, rates for these transitions become $r_{\mathrm{r},\frac{18}{14}}$ and $r_{\mathrm{r},\frac{27}{14}}$, respectively.
- Transitions C, A, I $\rightarrow$ E represent that an original tile can dissociate from the aggregate after the protective tile dissociated; these rates are $r_{\mathrm{r},2}$, $r_{\mathrm{r},1}$ and $r_{\mathrm{r},0}$, respectively.
- We must calculate the net forward rate r^* to evaluate rates for C $\rightarrow$ FC, A, and I $\rightarrow$ FI. This value is equivalent to the net transition rate between E and LC_1 r_f because a tile becomes fixed in place when it is buried by the next tile, which concurs with its neighbors.

[8] Branch migration is a random-walk process. Therefore, the reaction time t_bm depends on the base length and other parameters. The reaction time is difficult to estimate, but it does not strongly affect our result. For instance, even for very long t_bm like 1 /sec, no significant differences in error rate and growth speed are evident for $1 < \tau < 2$. For simplification, we assume that branch migration at S and W are synchronized.

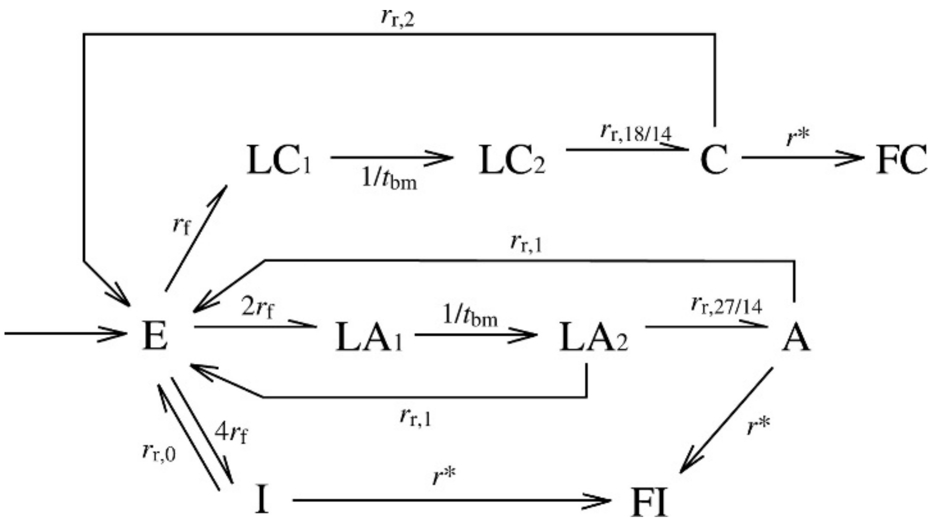


Fig. 8. The transition diagram for the LTM

Using the transition diagram, we can calculate the time development of each state. Let $p_i(t)$ be the probability of state (i) at time t. From the transition diagram in Fig. 8, we have the following differential equation:

$$\dot{\mathbf{p}}(t) = \begin{bmatrix} -7r_\mathrm{f} & 0 & 0 & 0 & r_{\mathrm{r},1} & r_{\mathrm{r},2} & r_{\mathrm{r},1} & r_{\mathrm{r},0} & 0 & 0 \\ r_\mathrm{f} & -\frac{1}{t_\mathrm{bm}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{t_\mathrm{bm}} & -r_{\mathrm{r},\frac{18}{14}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2r_\mathrm{f} & 0 & 0 & -\frac{1}{t_\mathrm{bm}} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{t_\mathrm{bm}} & -r_{\mathrm{r},1} - r_{\mathrm{r},\frac{27}{14}} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & r_{\mathrm{r},\frac{18}{14}} & 0 & 0 & -r_{\mathrm{r},2} - r^* & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & r_{\mathrm{r},\frac{27}{14}} & 0 & -r_{\mathrm{r},1} - r^* & 0 & 0 & 0 \\ 4r_\mathrm{f} & 0 & 0 & 0 & 0 & 0 & 0 & -r_{\mathrm{r},0} - r^* & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & r^* & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & r^* & r^* & 0 & 0 \end{bmatrix} \begin{bmatrix} p_\mathrm{E}(t) \\ p_{\mathrm{LC}_1}(t) \\ p_{\mathrm{LC}_2}(t) \\ p_{\mathrm{LA}_1}(t) \\ p_{\mathrm{LA}_2}(t) \\ p_\mathrm{C}(t) \\ p_\mathrm{A}(t) \\ p_\mathrm{I}(t) \\ p_\mathrm{FC}(t) \\ p_\mathrm{FI}(t) \end{bmatrix} \doteq M\mathbf{p}(t) \ , \qquad (3)$$

where $\mathbf{p}(0) = [1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]^T$.

For $t \to \infty$, probabilities of states must be zero except for FC and FI. Thus

$$\dot{\mathbf{p}}(\infty) = [-1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad p_{\mathrm{FC}}(\infty) \quad p_{\mathrm{FI}}(\infty)]^T = M\mathbf{p}(\infty) \ . \tag{4}$$

The error rate is derived as follows:

$$\epsilon = 1 - p_{\mathrm{FC}}(\infty) \ , \tag{5}$$

where p_{FC} is a solution of the above equation. (we can use software such as *Mathematica* to do this calculation.)

The growth speed is given by the net transition rate from E to C:

$$r_{\mathrm{g}} = \frac{1}{\dfrac{1}{r_{\mathrm{f}}} + t_{\mathrm{bm}} + \dfrac{1}{r_{\mathrm{r},\frac{18}{14}}}} - r_{\mathrm{r},2} \ . \tag{6}$$

Fig. 9 shows the error rate *vs.* growth speed of three models (OTM, 2×2 PTM and LTM) by this analysis. These plots are for G_{mc} at several values of G_{se} ($= 8.5$, 10.5, 12.5). Straight lines l_1, l_2, and l_3 represent the best performance region of these models, in which a low error rate and high growth speed are achieved. Slope of these lines indicate overall performance of each model. For OTM and 2×2 PTM, we have $r_{\mathrm{g}} \approx \beta\epsilon^2$ and $r_{\mathrm{g}} \approx \beta\epsilon$, respectively, where $\beta = 1 \times 10^5$ /M/sec is a constant.

For LTM, the relation between error rate and growth speed is

$$r_{\mathrm{g}} \approx \beta\epsilon \tag{7}$$

which means that LTM has similar performance to that of 2×2 PTM. In addition, LTM's performance curves overlap more with the optimal performance line l_3, implying that LTM has a low error rate for broader values of τ. In other words, LTM is more robust for variations of temperature and tile concentrations.

6 Conclusions

This paper proposed the Layered Tile Model (LTM). This model offers several merits:

- LTM provides an equivalent error rate and growth speed compared to 2×2 PTM.
- Using protective tiles, LTM can suppress unnecessary interactions among monomer and aggregates, which greatly hinder experimentation. Moreover, it suppresses random aggregates in the $0 < \tau < 1$ region.
- The number of sticky ends necessary for LTM is small. It requires about twice those of OTM. For any kind of tile set, we can design corresponding protective tiles by simple modification of the original tile set. A small number of necessary sticky ends is also advantageous to avoid spurious bindings among non-complementary sticky ends.

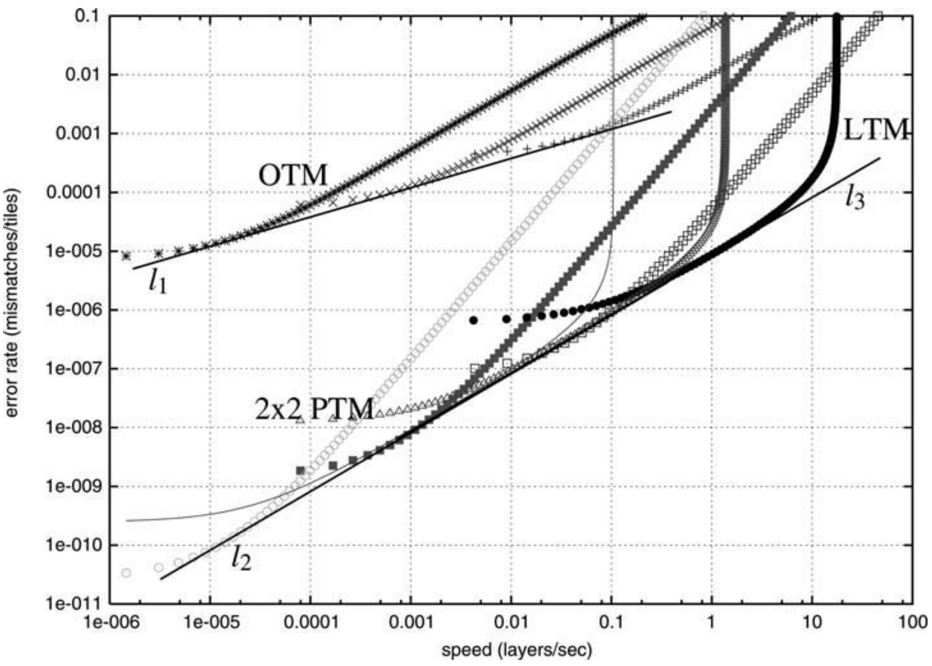


Fig. 9. Plots of ϵ *vs.* r_{g} of three models (OTM, 2×2 PTM and LTM) by analysis. Curved lines are plotted for G_{mc} at several values of G_{se} (= 8.5, 10.5, 12.5). Straight lines l_1, l_2, and l_3 represent the best performance of these models (l_2 overlaps l_3)

– LTM offers more robustness for changes of concentration and temperature. It shows good performance in a wider area of G_{se} and G_{mc} compared to OTM or 2×2 PTM. It is also important for practical experiments.

Further research is required for this model. The next step is implementation as DNA tiles. DNA sequences of original and protective tiles must be designed. Simple experiments are planned to verify protective tile functionality.

References

[1] Winfree, E., Yang, X., Seeman, N.C.: Universal computation via self-assembly of DNA: Some theory and experiments. In Landweber, L.F., Baum, E.B., eds.: DNA Based Computers II. Volume 44 of DIMACS., Providence, RI, American Mathematical Society (1996) 191–213
[2] Carbone, A., Seeman, N.C.: Circuits and programmable self-assembling DNA structures. Proc. Natl. Acad. Sci. USA **99** (2002) 12577–12582
[3] Kao, M.Y., Ramachandran, V.: DNA self-assembly for constructing 3D boxes. In Eades, P., Takaoka, T., eds.: ISAAC 2001. Volume 2223 of LNCS., Christchurch, New Zealand, Springer-Verlag (2001) 429–440

[4] Mao, C., LaBean, T.H., Reif, J.H., Seeman, N.C.: Logical computation using algorithmic self-assembly of DNA triple-crossover molecules. Nature **407** (2000) 493–496

[5] Rothemund, P.W.K., Winfree, E.: The program-size complexity of self-assembled squares. In: Symposium on Theory of Computing (STOC), Portland, Oregon, ACM (2000)

[6] Winfree, E., Liu, F., Wenzler, L.A., Seeman, N.C.: Design and self-assembly of two-dimensional DNA crystals. Nature **394** (1998) 539–544

[7] Yan, H., Feng, L., LaBean, T.H., Reif, J.H.: Parallel molecular computations of pair-wise XOR using DNA "String Tile" self-assembly. In Chen, J., Reif, J.H., eds.: DNA Based Computers 9. LNCS, Madison, Wisconsin, Springer-Verlag (2003) 97

[8] Yan, H., LaBean, T.H., Feng, L., Reif, J.H.: Directed nucleation assembly of DNA tile complexes for barcode-patterned lattices. Proc. Natl. Acad. Sci. USA **100** (2003) 8103–8108

[9] Winfree, E.: Simulations of computing by self-assembly. Technical Report CSTR:1998.22, Caltech (1998)

[10] Winfree, E., Bekbolatov, R.: Proofreading tile sets: Error correction for algorithmic self-assembly. In Chen, J., Reif, J.H., eds.: DNA Based Computers 9. LNCS, Madison, Wisconsin, Springer-Verlag (2003)

[11] Wang, H.: Proving theorems by pattern recognition. II. Bell System Technical Journal **40** (1961) 1–42

[12] Wang, H.: Dominoes and the AEA case of the decision problem. In Fox, J., ed.: Proceedings of the Symposium on the Mathematical Theory of Automata, Brooklyn, New York (1963) 23–55 Polytechnic Press.

[13] Fu, T.J., Seeman, N.C.: DNA double-crossover molecules. Biochemistry **32** (1993) 3211–3220

[14] Biswas, I., Yamamoto, A., Hsieh, P.: Branch migration through DNA sequence heterology. J. Mol. Biol. **279** (1998) 795–806

[15] Panyutin, I.G., Hsieh, P.: The kinetics of spontaneous DNA branch migration. Proc. Natl. Acad. Sci. USA **91** (1994) 2021–2025

[16] Turberfield, A.J., Mitchell, J.C.: DNA fuel for free-running nanomachines. Phys. Rev. Lett. **90** (2003) 118102

A Sticky End Design of Protective Tiles

We consider conditions in which protective tiles dissociate from original tiles only when they correctively associate with the aggregate.

Regarding rule tiles, let each sticky end strength of the protective tile be x_{r} and y_{r}, as shown in Figure 10a. First, x_{r} must be smaller than 1 because it must be replaced by branch migration between original tiles. y_{r} must be also smaller than 1 because the total strength of both N and E must be smaller than τ $(= 2 - \epsilon)$ when sticky ends of S and W dissociated. Secondly, total bonding strength of the protective tile must be larger than τ in the one-match case, namely $x_{\mathrm{r}} + 2y_{\mathrm{r}} \geq 2$. Thirdly, we add $y_{\mathrm{r}} < \frac{3}{4}$ to remove the protective tiles when two match (Fig. 10b).

Similarly, for the boundary tiles, we determine $0 < x_{\mathrm{b}} < 1$, $0 < y_{\mathrm{b}} < 1$, $x_{\mathrm{b}} + y_{\mathrm{r}} + y_{\mathrm{b}} \geq 2$, $y_{\mathrm{r}} + y_{\mathrm{b}} < \frac{3}{2}$.

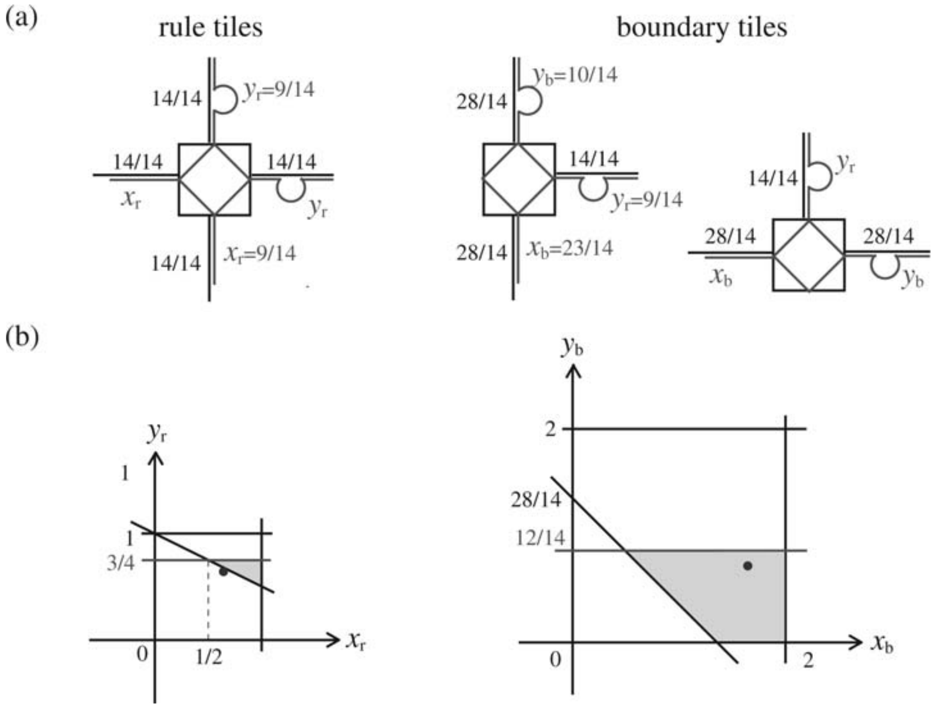


Fig. 10. (a) Layered tile structure. Sticky end strengths of protective tiles are x_{r}, y_{r}, x_{b}, and y_{b} (normalized by 1). (b) Yellow areas show possible strengths for the protective tiles. Blue points represent strengths ($x_{\mathrm{r}} = \frac{9}{14}$, $y_{\mathrm{r}} = \frac{9}{14}$, $x_{\mathrm{b}} = \frac{23}{14}$, $y_{\mathrm{b}} = \frac{10}{14}$), which we used in simulations and analyses

B Simulation Algorithm

The simulation algorithm is as follows: By Assumption iii in section 2, tiles are attached *one by one* to the aggregate. Initially, a corner tile (seed tile) is placed at the origin of the two-dimensional lattice. The rates of all possible reactions must be known to determine the next event, which is either to add a tile (*on-event*) or to remove a tile (*off-event*). The net on rate of *on-event* is $k_{\mathrm{on}} = mk_{\mathrm{f}}e^{-G_{\mathrm{mc}}}$, where m is the number of total empty sites adjacent to the aggregate. The net off rate of *off-event* is $k_{\mathrm{off}} = \sum_b k_{\mathrm{off},b}$, where $k_{\mathrm{off},b} = \sum_{ij \text{ s.t. } b_{ij}=b} k_{\mathrm{f}}e^{-b_{ij}G_{\mathrm{se}}}$. For each site $(i,\, j)$ belongs to the aggregate (except for the seed tile), the total strength b_{ij} is calculated. From k_{on} and k_{off}, the *on-event* probability is given by $\frac{k_{\mathrm{on}}}{k_{\mathrm{on}}+k_{\mathrm{off}}}$, and the *off-event* probability is $\frac{k_{\mathrm{off}}}{k_{\mathrm{on}}+k_{\mathrm{off}}}$. In the case of *on-event*, a tile (which is equally likely chosen from all kind of tiles) is added to one of the empty sites adjacent to the aggregate. In the case of the *off-event*, the probability exists that some tiles with b bonds dissociate from the aggregate. That probability is $\frac{k_{\mathrm{off},b}}{k_{\mathrm{off}}}$; all such sites present equal likelihood. Once the event is chosen, all rates must be recalculated to determine the next event.

Using Automated Reasoning Systems on Molecular Computing

Carmen Graciani Díaz and Mario J. Pérez-Jiménez

Research Group on Natural Computing,
Dpto. Ciencias de la Computación e Inteligencia Artificial,
Universidad de Sevilla (Spain)
{cgdiaz, marper}@us.es

Abstract. This paper is focused on the interplay between automated reasoning systems (as theoretical and formal devices to study the correctness of a program) and DNA computing (as practical devices to handle DNA strands to solve classical hard problems with laboratory techniques). To illustrate this work we have proven in the PVS proof checker, the correctness of a program, in a sticker based model for DNA computation, solving the pairwise disjoint families problem. Also we introduce the formalization of the Floyd–Hoare logic for imperative programs.

1 Introduction

One of the most active areas of research in Computer Science is the study and use of formal methods (applications of primarily discrete mathematics to software engineering problems). Its widely development and the complexity of interesting problems have given rise to automated reasoning. In this area, one of the main problems is the correctness [2]: developing specifications and proofs that ensures a program meets its specification. There is a previous work of formalization: expressing all definitions, theorems and proofs in a formal language without semantic ambiguity. This approximation has especial relevance in new computing paradigms such as the DNA based molecular computing. In many molecular models the data are *tubes* over an alphabet whose content encodes a collection of DNA strands. The operations considered are abstraction of different laboratory techniques to manipulate DNA strands.

This paper is organized as follows. It begins with a short presentation of the *Prototype Verification System (PVS)* and the sticker model. Then, how this model can be formalized in PVS, is briefly described. Section 4 introduces imperative programs and gives an overview of how we deal with them in PVS. Finally, as an example, a molecular solution of the pairwise disjoint families problem and a description of its formal verification obtained with PVS, is presented. The set of developed theories in PVS for this paper are available on the web at `http://www.cs.us.es/~cgdiaz/investigacion`.

2 The Prototype Verification System

The *Prototype Verification System (PVS)* is a proof checker based on higher–order logic where types have semantics according to Zermelo–Fraenkel set theory with the axiom of choice [8]. In such a logic we can quantify over functions which take functions as arguments and return them as values.

Specifications are organized into *theories*. They can be parameterized with semantic constructs (constant or types). Also they can import other theories. A *prelude* for certain standard theories is preloaded into the system. As an example we include in figure 1 the PVS theory `suc_finitas_def` which provides an alternative definition (to the type `finseq` given in the prelude) for sequences of a given length `n` for elements of a given type `V`.

```
suc_finitas_def[V: TYPE, n: nat]: THEORY
 BEGIN
% Finite sequence: S = {s_k}_k < n+1
  SUC_FINITAS: TYPE = [below[n] -> V]
  SF: TYPE = SUC_FINITAS
 END suc_finitas_def
```

Fig. 1. A PVS Theory

Before a theory may be used, it must be typechecked. The PVS typechecker analyzes the theory for semantic consistency and adds semantic information to the internal representation built by the parser. Since this is an undecidable process, the checks which cannot be resolved automatically are presented to the user as assertions called type–correctness conditions.

The PVS prover is goal–oriented. Goals are sequents consisting of antecedents and consequents, e.g. $A_1, \ldots, A_n \vdash B_1, \ldots, B_m$. The conjunction of the antecedents should imply the disjunction of consequents, i.e. $A_1 \wedge \cdots \wedge A_n \rightarrow B_1 \vee \cdots \vee B_m$. The proof starts with a goal of the form $\vdash B$, where B is the theorem to be proved. The user may type proof commands which either prove the current goal, or result in one or more new goals to prove. In this manner a proof tree is constructed. The original goal is proved when all leaves of the proof tree are recognized as true propositions. Basic proof commands can also be combined into strategies.

3 The Sticker Model: A Description Through PVS

The sticker model used in this paper was introduced by S. Roweis et al. [9] (this model is completely different from the *sticker systems* introduced by L. Kari et al in [6]). It is an abstract model of DNA based molecular computing with random access memory in the following sense: some operations could modify the structure of the DNA molecules and so the information codified by them changes during the execution.

In this model, a *memory strand* (a single stranded DNA molecule) N bases in length subdivided into `k` non–overlapping *regions* each M bases long is considered to represent a string of `k` bits. Each region is identified with exactly one bit position. Also, `k` different *sticker strands* (single stranded DNA molecule) each of them M bases long and complementary with one and only one of the `k` memory regions are considered. If a sticker is annealed to its matching region then the corresponding bit is *on*. Otherwise, it is *off*. A memory strand together with its associated stickers, if any, is called a *memory complex* and represent one bit string. In this sense we consider memory complexes as finite sequence of bits in PVS:

```
BITS: TYPE = {on, off}
MEMORY_COMPLEX: TYPE = finseq[BITS]
```

Associated with this definition we consider the application σ from $\mathbb{N}$ into {`on, off`} defined as follows (where σ_i is the i-th element of the bit sequent σ):

$$\sigma(i) = \begin{cases} \sigma_i & \text{if } i < \mathtt{k} \\ \mathtt{off} & \text{otherwise} \end{cases}$$

```
appl(sigma: MEMORY_COMPLEX, i: nat): BITS =
  IF i < sigma‘length THEN sigma‘seq(i) ELSE off ENDIF
```

Within sticker model a *tube* is a collection of memory complexes representing a multiset of bit strings. All memory strands (underlying each complex) in a tube are identical and each one has stickers annealed only at the required bit positions.

In PVS we consider a general concept: a multiset of memory complexes:

```
 GEN_TUBE: TYPE = MULTISETS[MEMORY_COMPLEX]
```

Then we restrict this definition to consider tubes containing only memory complexes of a given length, namely `k`.

```
MTUBE: TYPE =
  {T: GEN_TUBE | FORALL (sigma: MEMORY_COMPLEX):
                      ms_in(sigma, T) IMPLIES sigma‘length = k}
```

The following are the molecular operations on tubes used in the sticker model and the corresponding implementation in PVS.

– To *combine* two tubes producing a new one containing all the memory complexes from both tubes.

```
combine(T1, T2: GEN_TUBE): GEN_TUBE =
    LAMBDA (sigma: MEMORY_COMPLEX): T1(sigma) + T2(sigma)
```

– To *separate* the content of a tube into two new tubes, one containing all the memory complexes with a particular sticker annealed (a particular bit *on*) and the other all those with that region free (that bit *off*).

```
separate(T: GEN_TUBE, oi: nat): [GEN_TUBE, GEN_TUBE] =
    (LAMBDA (sigma: MEMORY_COMPLEX):
       IF appl(sigma, oi) = on THEN T(sigma) ELSE 0 ENDIF,
     LAMBDA (sigma: MEMORY_COMPLEX):
       IF appl(sigma, oi) = off THEN T(sigma) ELSE 0 ENDIF)
```

– To *turn on* (*set*) a particular region annealing the appropriate sticker on every complex in a tube (turning the corresponding bit to *on*).
– To *turn off* (*clear*) a particular region removing the appropriate sticker, if any, on every complex in a tube (turning the corresponding bit to *off*).
Previously to the implementation of these operations we define the concept of modifying in a memory complex, σ, a particular bit, i, to $\texttt{b} \in \{\texttt{on, off}\}$.

$$\sigma_i^{\texttt{b}} = \begin{cases} \{\sigma_0, \dots, \sigma_{i-1}, \texttt{b}, \sigma_{i+1}, \dots, \sigma_{\texttt{k}-1}\} & \text{if } i < \texttt{k} \\ \sigma & \text{otherwise} \end{cases}$$

```
turn(sigma: MEMORY_COMPLEX, i: nat, b: BITS): MEMORY_COMPLEX =
  IF i < sigma`length
    THEN sigma WITH [(seq) := sigma`seq WITH [(i) := b]]
  ELSE sigma ENDIF
```

From this we consider a general operation that changes, in all memory complexes present in a tube, T, a particular bit, i, to $\texttt{b} \in \{\texttt{on, off}\}$.

$$\texttt{Change(T, }i\texttt{, b) = } \{\!\{ \sigma_i^{\texttt{b}} \mid \sigma \in \texttt{T} \}\!\}$$

```
change(T: GEN_TUBE, i: nat, b: BITS): GEN_TUBE =
  LAMBDA (sigma: MEMORY_COMPLEX):
    IF i < sigma`length AND sigma`seq(i) = b
      THEN T(turn(sigma, i, off)) + T(turn(sigma, i, on))
    ELSIF i >= sigma`length THEN T(sigma) ELSE 0 ENDIF
```

The implementation of the turn operations (set and clear) are as follows:

$$\texttt{Set(T, i) = } \{\!\{ \sigma_i^{\texttt{on}} \mid \sigma \in \texttt{T} \}\!\} \qquad \texttt{Clear(T, i) = } \{\!\{ \sigma_i^{\texttt{off}} \mid \sigma \in \texttt{T} \}\!\}$$

```
set(T: GEN_TUBE, i: nat): GEN_TUBE = change(T, i, on)

clear(T: GEN_TUBE, i: nat): GEN_TUBE = change(T, i, off)
```

Also a *read* operation is considered. This operation determines if a tube is empty and otherwise selects a complex from the tube and produce the associated string of bits. To implement it we consider the special memory complex of length 0 as the answer when there is no elements in the tube.

```
read(T: GEN_TUBE): MEMORY_COMPLEX =
  IF EXISTS (gamma: MEMORY_COMPLEX): ms_in(gamma, T)
    THEN choose({sigma: MEMORY_COMPLEX | ms_in(sigma, T)})
  ELSE empty_seq ENDIF
```

Usually we express the use of those operations as assignments. For example,

$$\texttt{T} \longleftarrow \texttt{Combine}(\texttt{T}_1,\ \texttt{T}_2)$$

The interpretation of a program in the sticker model as a sequence of such operations has taken us to consider them as imperative programs.

4 Imperative Programs

Following [3] we consider an *imperative program* as a sequence, $\texttt{I}_1$ `@@` $\texttt{I}_2$ `@@` `...@@` $\texttt{I}_k$, of states transformers[1]. When such a program is executed on an initial state the first transformer is applied to it, the second is applied to the state obtained by the previous one and so on. A general work that shows how to deal in PVS with nontermination and nondeterministic state transformers can be found in [11].

A *state* is considered as a finite sequence of data in a given domain (we introduced the possibility of take a tuple of sequences over different domains to deal with elements of different nature). To access to the information stored in a state we have variables. Each variable is associated with a natural number in a one–to–one manner. The n–variable over a given state takes the value of the n–th element.

In general, a *term* is any function, `t`, that given a state, $\texttt{s} \in \texttt{S}$, produces an element over a given domain, `D`. Operations between elements of given domains are lifted to operations between terms using the function `l` we describe with an example. Suppose we have a binary operation $\texttt{op: D}_1\times\ \texttt{D}_2 \rightarrow \texttt{R}$. With `l` we obtain a binary operation `l(op)`, that given two terms $\texttt{t}_1\texttt{: S} \rightarrow \texttt{D}_1$ and $\texttt{t}_2\texttt{: S} \rightarrow \texttt{D}_2$ produces the term $\texttt{l(op)(t}_1\texttt{, t}_2\texttt{): S} \rightarrow \texttt{R}$ where

$$\texttt{l(op)(t}_1\texttt{, t}_2\texttt{)(s) = op(t}_1\texttt{(s), t}_2\texttt{(s))}$$

This function `l` is generalized to consider constants. Given $\texttt{c} \in \texttt{D}$, we obtain the term $\texttt{l(c): S} \rightarrow \texttt{D}$ such that `l(c)(s) = c`.

In [5], Hoare introduced the $\{\varphi\}$ `P` $\{\psi\}$ notation to describe the behaviour of a program `P`. Those expressions are called *specifications of partial correctness* and have the following meaning: If φ and ψ are some conditions over states the specification is true if whenever the program `P` is executed over a state verifying φ and it halts, then it produces a state verifying ψ. As the considered notion of program only consider total functions those expressions are, in fact, *specifications of total correctness.*

[1] In order to save space, we do not include in this section the corresponding PVS implementations, see [3] and [4] for more details.

To construct a formal proof of a specification of partial correctness we use the Floyd–Hoare logic, a set of axioms and inference rules. Next we introduce the ones used to construct the correctness proof in the following section.

– The *consequence rule*:

$$\boxed{\dfrac{\varphi \to \varphi',\ \{\varphi'\}\ \mathtt{S}\ \{\psi'\},\ \psi' \to \psi}{\{\varphi\}\ \mathtt{S}\ \{\psi\}}}$$

– The *assignment instruction*, $\mathtt{X} \longleftarrow \mathtt{t}$ (we denote $\longleftarrow$ by `<<` in PVS) is a program that over a state `s` produces the state `s[t(s)/X]`, resulting from `s` after the substitution of the associated value for the variable `X` by `t(s)`.
  The *assignment axiom* is

$$\boxed{\{\varphi\mathtt{[t/X]}\}\ \mathtt{X} \longleftarrow \mathtt{t}\ \{\varphi\}}$$

  where φ`[t/X](s) =` φ`(s[t(s)/X])`.
– The *compose rule*:

$$\boxed{\dfrac{\{\varphi\}\ \mathtt{S}_1\ \{\psi\},\ \{\psi\}\ \mathtt{S}_2\ \{\phi\}}{\{\varphi\}\ \mathtt{S}_1\ \mathtt{@@}\ \mathtt{S}_2\ \{\phi\}}}$$

– Given a program `P`, the following form

```
for X from 0 to t-1 do
    P
end for
```

  (we write it `loop(X, t, P)` for short) has the following meaning

$$\mathtt{X} \longleftarrow \mathtt{0\ @@\ P\ @@\ \ldots @@\ X} \longleftarrow \mathtt{t\text{-}1\ @@\ P}$$

  The loop rule is

$$\boxed{\begin{array}{c}\dfrac{\{\varphi \wedge \mathtt{X < t}\}\ \mathtt{P}\ \{\varphi\mathtt{[X+1/X]}\}}{\{\varphi\mathtt{[0/X]}\}\ \mathtt{loop(X,\ t,\ P)}\ \{\varphi\mathtt{[t/X]}\}}\\ \text{no assignment to } \mathtt{X} \text{ or variables occurring in } \mathtt{t} \text{ is used in } \mathtt{P}\end{array}}$$

4.1 First Order Logic

To express conditions over states we consider a first order logic whose set of terms, `TERM`, is the inductive closure of the union of the set of variables mentioned above and the set of lifted constants under the constructors `l(op)` for every function `op`. The set of atomic formulas is the inductive closure of the pair of sets `TERM` and {`l(p)| p boolean constant`}, under the constructors `l(op)` for every predicate `op`.

Previous to the definition of the set of formulas we need the concept of the state, `s[d/X]`, resulting from a given one, `s`, after the substitution of the associated value for a variable `X` by `d`. That is, `s[d/X]` is a state such that for any other variable different from X it has the same associated value than s and for X it has d as the associated value.

The set of formulas is the inductive closure of the set of atomic formulas under the constructors l(∧), l(∨), l(¬), foreach and exists, where

$$\texttt{foreach(X, }\varphi\texttt{): S} \to \texttt{bool} \text{ such that } \texttt{foreach(X, }\varphi\texttt{)(s)} \equiv \forall\ \texttt{d}\ (\varphi\texttt{(s[d/X]))}$$

$$\texttt{exists(X, }\varphi\texttt{): S} \to \texttt{bool} \text{ such that } \texttt{exists(X, }\varphi\texttt{)(s)} \equiv \exists\ \texttt{d}\ (\varphi\texttt{(s[d/X]))}$$

5 The Pairwise Disjoint Families Problem

Let us consider the following problem:

Let A = {0, ..., p-1}. Let $\mathcal{F}$ = {B_0, ..., B_{q-1}} a finite family of subsets of A. To determine all the ordered pairs ($\mathcal{F}$', $\bigcup\mathcal{F}$'), where $\mathcal{F}$' is a subfamily of $\mathcal{F}$ and its elements are pairwise disjoint.

To solve this problem in the sticker model we consider as initial tube T_0, a (p+q, q)–library (a tube containing, at least, a copy of any memory complex with p+q regions and the p last regions deactivated). The first q bits represent a subfamily of $\mathcal{F}$. Given a memory complex with p+q regions, σ, we consider that it codifies an ordered pair ($\mathcal{F}_\sigma$, A_σ), where $\mathcal{F}_\sigma$ is the subfamily $\{B_j \mid \sigma(j) = \text{on}\}$ of $\mathcal{F}$ and A_σ is the subset $\{j \mid \sigma(j+q) = \text{on}\}$ of A.

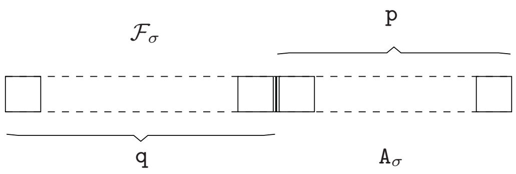


Fig. 2. Memory complex with (p+q) regions

The following is a program in the sticker model that solves the pairwise disjoint families problem (where b^i_j is the j-th element of B_i, the i-th subset of $\mathcal{F}$, and r_i is its size; that is, $B_i = \{b^i_0, \ldots, b^i_{r_i-1}\} \in \mathcal{F}$).

Note: Each instruction is labeled in order to make references.

```
              Procedure Disjoint
              INPUT: A family $\mathcal{F}$ of A subsets
$I_1$                 for I ⟵ 0 to q-1 do
$L_1$                        (T*, T-) ⟵ Separate(T, I) @@
$L_2$                        for J ⟵ 0 to $r_I$-1 do
$l_1$                               (T+, T'-) ⟵ Separate(T*, $b^I_J$+q) @@
$l_2$                               T* ⟵ Set(T'-, $b^I_J$+q)
                             end for @@
$L_3$                        T ⟵ Combine(T*, T-)
                      end for
```

The following PVS expression implements the program:

```
disjoint(F: (FAMILY(p, q))): program =
LET eB = l(elemF(p,q,F)) IN
loop(VI, q,
     assig2((VTast, VTn), l(separate)(VT, VI)) @@
     loop(VJ, l(tam(p,q,F))(VI),
          assig2((VTm, VTnn), l(separate)(VTast, eB(VI, VJ) + l(q))) @@
          (VTast << l(set)(VTnn, eB(VI, VJ) + l(q)))) @@
     (VT << l(combine)(VTast, VTn)))

assig2(PT: [V1, V1], Pt: [term1, term1]): program =
      (PT‘1 << Pt‘1) @@ (PT‘2 << Pt‘2)
```

In order to stablish the correctness of this program we consider the formula:

$$\Theta_{\mathcal{F}}(\mathtt{T}) \equiv \forall\tau\ (\tau \in \mathtt{T} \rightarrow \forall\ \mathtt{i}_1 < \mathtt{i}_2 < \mathtt{q}\ (\tau(\mathtt{i}_1) = \tau(\mathtt{i}_2) = \mathtt{on} \rightarrow \mathtt{B}_{\mathtt{i}_1} \cap \mathtt{B}_{\mathtt{i}_2} = \emptyset))$$

expressing that the memory complexes of a given tube codifies a subfamily of $\mathcal{F}$ whose elements are pairwise disjoint.

```
correc_disjoint(F: (FAMILY(p, q)))(T: MTUBE[p + q]): bool =
  FORALL (tau: MEMORY_COMPLEX): (ms_in(tau, T) IMPLIES
      (FORALL (i1, i2: below[q]):
        (appl(tau, i1) = on AND appl(tau, i2) = on AND i1 < i2 IMPLIES
          disj(F‘seq(i1), F‘seq(i2)))))
```

The following specification stablish the correctness of the program (where `library?[p+q](q)` is a predicate over tubes characterizing a `(p+q, q)`–library):

$$\{\mathtt{library?[p+q](q)(T)}\}\ \mathtt{disjoint}(\mathcal{F})\ \{\Theta_{\mathcal{F}}(\mathtt{T})\}$$

To prove this specification we will use two formulas θ and δ, that will be invariants of the main loop ($\mathtt{I}_1$) and inner loop ($\mathtt{L}_2$), respectively. For these formulas we prove the following results:

1. $\mathtt{library?[p+q](q)(T)} \rightarrow \theta\mathtt{[0/I]}$
2. $\theta\mathtt{[q/I]} \rightarrow \Theta_{\mathcal{F}}(\mathtt{T})$
3. $\theta \wedge \mathtt{I} < \mathtt{q} \rightarrow \delta\mathtt{[0/J][+(T,I)/T*][-(T,I)/T-]}$
4. $\delta\mathtt{[r_I/J]} \rightarrow \theta\mathtt{[I+1/I][T*} \cup \mathtt{T-/T]}$
5. $\delta \wedge \mathtt{J} < \mathtt{r_I} \rightarrow$
   $\delta\mathtt{[J+1/J][Set(T'-,\ b^I_J+q)/T*][-(T*,\ b^I_J+q)/T'-][+(T*,\ b^I_J+q)/T+]}$

From those results and using the appropriate axioms and inference rules from Floyd–Hoare logic we prove the following specifications:

– $\{\delta*\}\ \mathtt{l}_1\ \mathtt{@@}\ \mathtt{l}_2\ \{\delta\mathtt{[J+1/J]}\}$ where $\delta*$ is the formula
  $\delta\mathtt{[J+1/J][Set(T'-,\ b^I_J+q)/T*][-(T*,\ b^I_J+q)/T'-][+(T*,\ b^I_J+q)/T+]}$
  (using the assignment axiom and the compose rule).

– $\{\delta \wedge \texttt{J} < \texttt{r}_\texttt{I}\}$ $\texttt{l}_1$ @@ $\texttt{l}_2$ $\{\delta\texttt{[J+1/J]}\}$ (using 5 and the consequence rule).
– $\{\delta\texttt{[0/J]}\}$ $\texttt{L}_2$ $\{\delta\texttt{[r}_\texttt{I}\texttt{/J]}\}$ (using the loop for rule).
– $\{\delta\texttt{[0/J]}\}$ $\texttt{L}_2$ $\{\theta\texttt{[I+1/I][T*} \cup \texttt{T-/T]}\}$ (using 4 and the consequence rule).
– $\{\delta\texttt{[0/j][+(T,I)/T*][-(T,I)/T-]}\}$ $\texttt{L}_1$ @@ $\texttt{L}_2$ @@ $\texttt{L}_3$ $\{\theta\texttt{[I+1/I]}\}$ (with the assignment axiom and the compose rule).
– $\{\theta \wedge \texttt{I} < \texttt{q}\}$ $\texttt{L}_1$ @@ $\texttt{L}_2$ @@ $\texttt{L}_3$ $\{\theta\texttt{[I+1/I]}\}$ (with 3 and the consequence rule).
– $\{\theta\texttt{[0/I]}\}$ $\texttt{I}_1$ $\{\theta\texttt{[q/I]}\}\}$ (using the loop for rule).
– {library?[p+q](q)(T)} disjoint($\mathcal{F}$) $\{\Theta_\mathcal{F}\texttt{(T)}\}$ (using 1, 2 and the consequence rule).

The used formulas, θ and δ, are the following:

$$\theta \equiv \theta_D\texttt{(T,I)} \wedge \theta_R\texttt{(T,I)} \wedge \texttt{(I=0} \rightarrow \texttt{library?[p+q](q)(T)])}$$

$$\delta \equiv \delta_D\texttt{(T*,I,J)} \wedge \theta_D\texttt{(T-,I)} \wedge \texttt{carac(T-,I)} \wedge \delta_R\texttt{(T*,I,J)} \wedge \theta_R\texttt{(T-,I)}$$

where

– $\theta_D\texttt{(T,I)}$ is the formula:
$\texttt{I} \leq \texttt{q} \rightarrow \forall\tau\ (\tau \in \texttt{T} \rightarrow \forall i_1 < i_2 < \texttt{I}\ (\tau(i_1) = \tau(i_2) = \texttt{on} \rightarrow \texttt{B}_{i_1} \cap \texttt{B}_{i_2} = \emptyset))$
expressing that for each memory complex τ of a tube T, the elements of the subfamily $\mathcal{F}_\tau^\texttt{I} = \{\texttt{B}_i |\ \texttt{i} < \texttt{I} \wedge \tau(i) = \texttt{on}\}$ are pairwise disjoint.
– $\theta_R\texttt{(T, I)}$ is the formula:
$$\begin{aligned}&\texttt{I} \leq \texttt{q} \rightarrow \forall\tau\ (\tau \in \texttt{T} \rightarrow \\ &\qquad \forall k < \texttt{I}\ (\tau(k) = \texttt{on} \rightarrow \texttt{B}_k\texttt{+q} \subseteq \tau) \wedge \\ &\qquad \forall s < \texttt{p}\ (\tau(s\texttt{+q}) = \texttt{on} \rightarrow \exists k < \texttt{I}\ (\tau(k) = \texttt{on} \wedge s \in \texttt{B}_k)))\end{aligned}$$
that is, for each memory complex, τ, of a tube T, we have $\bigcup \mathcal{F}_\tau^\texttt{I} = \texttt{A}_\tau$.
– $\delta_D\texttt{(T,I,J)}$ is the formula:
$$\begin{aligned}&\texttt{I} < \texttt{q} \wedge \texttt{J} \leq r_\texttt{I} \rightarrow \\ &\quad \forall\tau\ (\tau \in \texttt{T} \rightarrow (\tau(I) = \texttt{on} \rightarrow \forall i_1 < \texttt{I}\ (\tau(i_1) = \texttt{on} \rightarrow \texttt{B}_{i_1} \cap \texttt{B}_\texttt{I}^\texttt{J} = \emptyset)) \wedge \\ &\qquad\qquad \forall i_1 < i_2 < \texttt{I}\ (\tau(i_1) = \tau(i_2) = \texttt{on} \rightarrow \texttt{B}_{i_1} \cap \texttt{B}_{i_2} = \emptyset))\end{aligned}$$
expressing that for each memory complex, τ, in a tube T, if $\texttt{B}_\texttt{I} \in \mathcal{F}_\tau^{\texttt{I+1}}$, then the set $\texttt{B}_\texttt{I}^\texttt{J}$ (compose by the first J elements of $\texttt{B}_\texttt{I}$) is disjoint with the elements of the subfamily $\mathcal{F}_\tau^\texttt{I}$; and that the elements of the subfamily $\mathcal{F}_\tau^\texttt{I}$ are pairwise disjoint.
– $\delta_R\texttt{(T,I,J)}$ is the formula
$$\begin{aligned}&\texttt{I} < \texttt{q} \wedge \texttt{j} \leq r_\texttt{I} \rightarrow \\ &\quad \forall\tau\ (\tau \in \texttt{T} \rightarrow (\tau\texttt{(I)} = \texttt{on} \wedge \forall k < \texttt{I}\ (\tau(k) = \texttt{on} \rightarrow \texttt{B}_k\texttt{+q} \subseteq \tau) \wedge \texttt{B}_\texttt{I}^\texttt{J}\texttt{+q} \subseteq \tau \wedge \\ &\qquad \forall s < \texttt{p}\ (\tau(s\texttt{+q}) = \texttt{on} \rightarrow \exists k < \texttt{I}\ ((\tau(k) = \texttt{on} \wedge s \in \texttt{B}_k) \vee s \in \texttt{B}_\texttt{I}^\texttt{J}))))\end{aligned}$$
expressing that for each memory complex, τ, in a tube T, we have
$$(\bigcup \mathcal{F}_\tau^\texttt{I}) \cup \texttt{B}_\texttt{I}^\texttt{J} = \texttt{A}_\tau$$
– $\texttt{carac(T, I)} \equiv \forall\tau\ (\tau \in \texttt{T} \rightarrow \tau\texttt{(I)} = \texttt{off)}$.
This formula characterizes the contents of the second tube obtained after the use of the Separate operation.

6 Conclusions

A great part of our work within molecular computing is related to the formalization of the different models that have appeared. During this effort we have drawn the conclusion that using formal notations does not ensure us that specifications will be correct. They still need to be validated by permanent reviews but, on the other hand, they support formal deduction; thus, reviews can be supplemented by mechanically checked analysis. One advantage of PVS is that it has sets and functions as types and that it is based on a higher–order logic so we gain expressiveness.

To develop the work presented we have provided not only an implementation of the sticker model in PVS. All the elements necessary to represent problems over finite sets of natural numbers has been described in the system. Also we have proved some usual properties over them and plan to complete this work for general purpose. Most of the proofs constructed with the system have been obtained using the basic commands and a previously elaborated hand written proof. This effort shows the utility of the system as a verification tool.

References

1. L. M. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266:1021–1024, November 1994.
2. R. S. Boyer and J S. Moore. *The correctness problem in computer science.* Academic Press, 1981.
3. P. Y Gloess. Imperative program verification in PVS. `http://www.labri.fr/Perso/~gloess/imperative/` (1999).
4. C. Graciani Díaz. *Especificación y verificación de programas moleculares en PVS.* Doctoral Thesis, University of Seville (2003).
5. C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10), 576–583, 1969.
6. Lila Kari, Gheorghe Paun, Grzegorz Rozenberg, Arto Salomaa, and S. Yu. DNA computing, sticker systems and universality. *Acta Informatica*, 35:401–420, 1998.
7. S. Owre, N. Shankar and J. Rushby *The PVS specification and verification system.* `pvs.csl.sri.com`
8. S. Owre and N. Shankar *The formal semantics of PVS.* Technical Report SRI-CSL-97-2, Computer Science Laboratory, SRI International, Menlo Park, CA, August 1997.
9. Roweis, S.; Winfree, E.; Burgoyne, R.; Chelyapov, N. V.; Goodman, M. F.; Rothemund, P. W. K.; Adleman, L. M. A sticker based model for DNA computation. Landweber, L.; Baum, E., eds *DNA Based Computers II*, DIMACS: Series in Discrete Mathematics and Theoretical Computer Science, 44, 1–27. American Mathematical Society (1999).
10. Sancho, F. *Verificación de programas en modelos de computación no convencionales.* Doctoral Thesis, University of Seville (2002).
11. H. Pfeifer, A. Dold, F. W. v. Henke, and H. Rueß. Mechanized Semantics of Simple Imperative Programming Constructs. Ulmer Informatik-Berichte 96-11, Universität Ulm, Fakultät für Informatik, 1996.

Parallelism in Gene Assembly

Tero Harju[1,3], Chang Li[2,3], Ion Petre[2,3], and Grzegorz Rozenberg[4]

[1] Department of Mathematics, University of Turku,
Turku 20014 Finland
harju@utu.fi

[2] Department of Computer Science, Åbo Akademi University,
Turku 20520 Finland
{lchang, ipetre}@abo.fi

[3] Turku Centre for Computer Science, Turku 20520, Finland

[4] Leiden Institute for Advanced Computer Science, Leiden University,
Niels Bohrweg 1, 2333 CA Leiden, the Netherlands and
Department of Computer Science, University of Colorado at Boulder,
Boulder, Co 80309-0347, USA
rozenber@liacs.nl

Abstract. The process of gene assembly in ciliates, an ancient group of organisms, is one of the most complex instances of DNA manipulation known in any organisms. This process is fascinating from the computational point of view, with ciliates even using the linked list data structure. Three molecular operations (ld, hi, and dlad) have been postulated for the gene assembly process. We initiate here the study of parallelism of this process by investigating several natural questions, such as: when can a number of operations be applied in parallel to a gene pattern, or how many steps are needed to assemble in parallel a micronuclear gene. We believe that the study of parallelism contributes to a better understanding of the nature of gene assembly, and in particular it provides a new insight in the complexity of this process.

1 Introduction

The ciliates (ciliated protozoa) are an ancient and diverse group of unicellular organisms. Their diversity can be appreciated by comparing their genomic sequences – some ciliate types differ genetically more than humans differ from fruit flies! A unique feature of the ciliates is their nuclear dualism: each ciliate possesses two kinds of nuclei in the same cell, a *micronucleus* and a *macronucleus*, see [9], [10], and [11]. The micronucleus is a germline nucleus and has no known function in the growth or the division of the cell. All RNA transcripts are provided by the macronucleus – the somatic nucleus. The two types of nuclei are however interrelated: at some stage, in the process of sexual reproduction, the genome of the micronucleus develops into the genome of the macronucleus, in a process called *gene assembly*. What makes this process unusual is the sophisticated rearrangement that ciliates, and in particular the *Stichotrichs* have engineered in the DNA sequence of their micronuclear genome. Thus, while genes in the macronucleus are contiguous sequences, placed (with very few exceptions) on their own short DNA molecules, the DNA in micronucleus is organized in long molecules, with genes occurring individually or in groups, separated by long stretches of non-coding DNA. The genes in the micronucleus are broken into pieces called MDSs, separated by non-coding segments called IESs. Moreover, the MDSs may be scrambled, i.e., the sequence of MDSs is permuted, with some MDSs being inverted. During gene assembly, the IESs are excised and MDSs are ligated to form transcriptionally competent macronuclear genes.

The gene assembly process is highly interesting from the computational point of view. One of the amazing features of this process is that ciliates apparently know *linked lists* and use them in an elegant pattern matching mechanism.

Three molecular operations, ld, hi, and dlad, have been postulated in [8] and [12] for the gene assembly process – they were successfully used to provide a uniform explanation for all known experimental data. The gene structure and the operations themselves have been modelled and formally investigated on three levels of abstraction based on permutations, strings, and graphs see [1], [6], [8], and [12]. This line of research has already answered a number of natural questions, such as the assembly power of these operations, invariants of the gene assembly, or micronuclear gene patterns that can be assembled using a subset of operations, see, e.g., [2], [3], [5], and [7]. We refer to the recent monograph [4] for a comprehensive treatment of this research area.

In our research so far, the process of gene assembly has been mostly considered as a *sequence* of folding and recombination operations. While this approach was adequate for the type of research questions that have been considered, in order to gain more insight into the gene assembly process, a more general *parallel* application of molecular operations must be investigated – *parallelism* is a natural phenomenon in biomolecular processes. In this paper we initiate a systematic study of parallelism in our model for gene assembly. Intuitively, a number of operations can be applied in parallel to a gene pattern if each operation's applicability is independent of the other's. In other words, a set of operations can be applied in parallel to a gene pattern if and only if they can be (sequentially) applied to that pattern in any order – this is consistent with how *concurrency* and *parallelism* are usually defined in Computer Science.

Our notion of parallelism naturally leads to a new measure of complexity for the gene assembly process, given by the minimal number of steps required to assemble a gene in parallel. E.g., micronuclear genes having the MDSs in the orthodox order, such as *C2* and β *TP* in *S. nova*, should be intuitively equally easy to assemble. This is indeed the case, as we discuss in this paper: the signed graphs associated to such genes can be reduced to the empty graph (the abstraction of the completion of the gene assembly process for signed graphs) in one parallel step. This clearly leads to another question: how many steps are needed in general to reduce a signed graph (or to assemble a gene pattern)? We conjecture a stunning answer to this question: any negative graph can be assembled in parallel in at most two steps! Note however that we assume here maximal parallelism: any operation that can be applied in a given step of the reduction must be applied at that stage. Whether or not ciliates actually operate in this way is clearly a different question that can be answered only through well-designed laboratory experiments.

2 Operations for Gene Assembly

Three molecular operations were postulated in [8] and [12] for the gene assembly in ciliates. We only show here in Fig. 1-3 the foldings required by each operation and the recombinations that take place in each case. We refer to [4] for a detailed discussion.

The central role in gene assembly is played by characteristic short sequences at the ends of MDSs, called *pointers* – the pointer in the end of an MDS M coincides (as a nucleotide sequence) with the pointer in the beginning of the MDS succeeding M in the macronuclear gene. Each micronuclear gene and its intermediary successors in the gene assembly process can be thus described by *signed permutations* (denoting the sequence and the orientation of the MDSs), *signed double occurrence strings* (denoting the sequence and the orientation of the pointers), and *signed graphs* (denoting the overlap of the pointers). E.g., the signed graph associated to the micronuclear gene *actin I* in *S. nova* is given in Fig. 4. We refer to [4] for many other examples.

Surprisingly enough, it has been proved in [1] and [6] that the information given by the overlap relations among pointers is sufficient for analyzing the whole

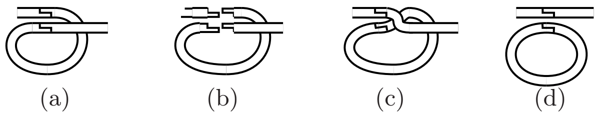


Fig. 1. Illustration of the ld molecular operation

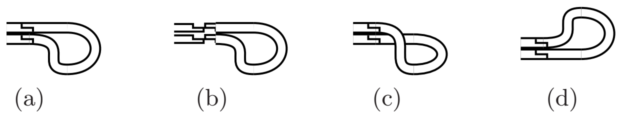


Fig. 2. Illustration of the hi molecular operation

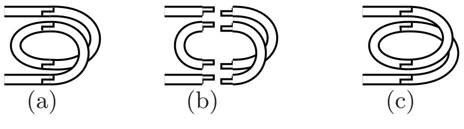


Fig. 3. Illustration of the dlad molecular operation

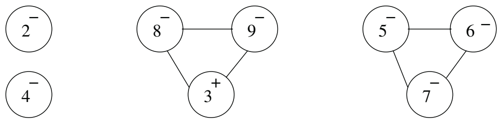


Fig. 4. The signed overlap graph associated to he micronuclear gene encoding the actin protein in *S. nova*

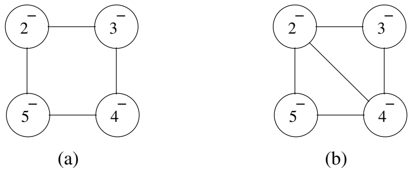


Fig. 5. (a) The square C_4; (b) The diamond D_4

process of gene assembly. We refer to [1], [6], and [4] for all details concerning the various levels of abstraction and for the methodology of model forming. We focus in this paper on the graph level. We recall below some basic definitions related to signed graphs – we refer to [13] for more details on Graph Theory.

A *signed graph* G is a structure $G = (V, E, \sigma)$ where (V, E) is a nondirected graph and $\sigma : V \to \{+, -\}$ is a vertex-labelling function. G is called the *empty graph*, denoted $\emptyset$, if $V = \emptyset$. We denote an edge between vertices u, v as uv – since our graphs are nondirected, we have $uv = vu$ for all edges $uv \in E$. A vertex $v \in V$ is *positive* (*negative*, resp.) if $\sigma(v) = +$ ($\sigma(v) = -$, resp.) Denote $V^+ = \{v \in V \mid \sigma(v) = +\}$ and $V^- = V \setminus V^+$, and let G^+ (G^-, resp.) be the signed subgraph of G induced by V^+ (V^-, resp.). For a vertex $p \in V$ we will also write $p \in G$. We say that G is *all-negative* (*all-positive*, resp.) if $V = V^-$ ($V = V^+$, resp.). The *neighborhood* of $v \in V$ is $N_G(v) = \{u \in V \mid uv \in E\}$; v is *isolated* if $N_G(v) = \emptyset$. G is *discrete* if all its vertices are isolated. G is called a *clique* if $E = \{uv \mid u, v \in V, u \neq v\}$. The *complement* of G is the signed graph (V, E', σ'), where $uv \in E'$ if and only $uv \notin E$, for all $u \neq v$, and $\sigma'(w) = +$ if and only if $\sigma(w) = -$, for any $w \in V$.

For two signed graphs G_1, G_2, with $G_i = (V_i, E_i, \sigma_i)$, for $i = 1, 2$ and $V_1 \cap V_2 = \emptyset$, we denote by $G_1 \oplus G_2$ the *disjoint union* of G_1 and G_2, i.e., the graph (V_3, E_3, σ_3) with $V_3 = V_1 \cup V_2$, $E_3 = E_1 \cup E_2$, $\sigma_3(u) = \sigma_1(u)$ if $u \in V_1$ and $\sigma_3(u) = \sigma_2(u)$ if $u \in V_2$. We denote by $G_1 \otimes G_2$ the *complete connection* of graphs G_1 and G_2, i.e., the signed graph (V_4, E_4, σ_4) with $V_4 = V_1 \cup V_2$, $E_4 = E_1 \cup E_2 \cup \{uv \mid u \in V_1, v \in V_2\}$, $\sigma_4 = \sigma_3$. A signed graph G is called *complete tripartite* if there are discrete graphs G_1, G_2, G_3 such that $G = G_1 \otimes G_2 \otimes G_3$. A signed graph (V, E, σ) is called a *square*, denoted C_4 (*diamond*, resp., denoted D_4) if it is isomorphic to the graph in Fig. 5(a) (Fig. 5(b), resp.).

The process of gene assembly is modelled on signed graphs by *reduction strategies*: assembling the gene corresponds to reducing its associated signed graph to the empty graph using one of the three rules (gnr, gpr, gdr) defined below. We first introduce some preliminary notations.

Let $G = (V, E, \sigma)$ be a signed graph and $S \subseteq V$. We say that the signed graph $G' = (V, E', \sigma')$ is obtained from G by *complementing* the set of vertices S if G' results from G by replacing the subgraph induced by S with its complement; G' is denoted by $\mathsf{com}_S(G)$. Moreover, if S is the neighborhood $N_G(v)$ of $v \in V$, then we get the *local complement* $\mathsf{loc}_v(G)$ at v: $\mathsf{loc}_v(G) = \mathsf{com}_{N_G(v)}(G)$. For a vertex $u \in V$, we denote by $G - u$ the subgraph of G induced by $V \setminus \{u\}$. For disjoint sets of vertices $S_1, \ldots, S_k \subseteq V$, we denote by $\mathsf{switch}_{S_1,\ldots,S_k}(G)$ the graph $G' = (V, E', \sigma)$ where for any two vertices u, v, if $u \in S_i$, $v \in S_j$, $i \neq j$, then $uv \in E'$ if and only if $uv \notin E$, otherwise, $uv \in E'$ if and only if $uv \in E$.

The molecular operations ld, hi, and dlad are modelled on signed graphs by the rules gnr, gpr, and gdr defined below. Let G be a signed graph.

The *graph negative rule* gnr_p for a vertex p is applicable to G if $p \in G^-$ is isolated. The result $\mathsf{gnr}_p(G)$ is the signed graph $\mathsf{gnr}_p(G) = G - p$. The *domain* of gnr_p is $\mathsf{dom}(\mathsf{gnr}_p) = \{p\}$. Let $\mathsf{Gnr} = \{\mathsf{gnr}_p \mid p \geq 1\}$ be the set of all graph negative rules on signed graphs.

The *graph positive rule* gpr_p for a vertex p is applicable to G if $p \in G^+$. The result $\mathsf{gpr}_p(G)$ is the signed graph $\mathsf{gpr}_p(G) = \mathsf{loc}_p(G) - p$. The *domain* of gpr_p is $\mathsf{dom}(\mathsf{gpr}_p) = \{p\}$. Let $\mathsf{Gpr} = \{\mathsf{gpr}_p \mid p \geq 1\}$ be the set of all graph positive rules on signed graphs.

The *graph double rule* $\mathsf{gdr}_{p,q}$ for two different vertices p and q is applicable to G if $p, q \in G^-$ are adjacent. Denoting $N_1 = N_G(p) \setminus N_G(q)$, $N_2 = N_G(p) \cap N_G(q)$, $N_3 = N_G(q) \setminus N_G(p)$, the result $\mathsf{gdr}_{p,q}(g)$ is the signed graph $\mathsf{gdr}_{p,q}(G) = \mathsf{switch}_{N_1,N_2,N_3}(G) - p - q$. The *domain* of $\mathsf{gdr}_{p,q}$ is $\mathsf{dom}(\mathsf{gdr}_{p,q}) = \{p, q\}$. Let $\mathsf{Gdr} = \{\mathsf{gdr}_{p,q} \mid p, q \geq 1\}$ be the set of all graph double rules on signed graphs. It is straightforward to prove that $\mathsf{gdr}_{p,q}(G) = \mathsf{gpr}_p(\mathsf{gpr}_q(\mathsf{loc}_p(G)))$, see [4], Lemma 11.3 for its counterpart result for signed double occurrence strings.

For a signed graph G and a set of operations $\{\varphi_1, \varphi_2, \ldots, \varphi_n\} \subseteq \mathsf{Gnr} \cup \mathsf{Gpr} \cup \mathsf{Gdr}$, we say that $\varphi = \varphi_1 \circ \varphi_2 \circ \cdots \circ \varphi_n$ is a *strategy* for G if $\varphi(G) = \emptyset$.

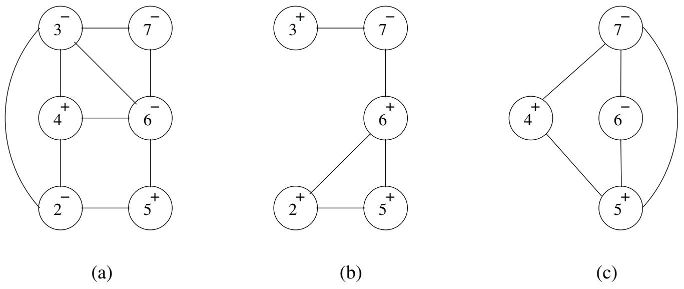


Fig. 6. (a) A signed graph G; (b) The graph $\mathsf{gpr}_4(G)$; (c) The graph $\mathsf{gdr}_{2,3}(G)$

Example 1. Consider the signed graph G illustrated in Fig. 6(a). Then gpr_4 and $\mathsf{gdr}_{2,3}$ are applicable to G, $\mathsf{gpr}_4(G)$ is shown in Fig.6(b), and $\mathsf{gdr}_{2,3}(G)$ is shown in Fig. 6(c).

3 Parallelism in Gene Assembly

In this section we formalize the notion of parallelism for signed graphs.

Definition 1. *Let $S \subseteq \mathsf{Gnr} \cup \mathsf{Gpr} \cup \mathsf{Gdr}$ be a set of rules and let $G = (V, E, \sigma)$ be a signed graph. We say that the rules in S can be applied in parallel to G if for any ordering $\varphi_1, \varphi_2, \ldots, \varphi_k$ of S, the composition $\varphi_k \circ \cdots \circ \varphi_1$ is applicable to G. In particular, two rules $\varphi, \psi \in \mathsf{Gnr} \cup \mathsf{Gpr} \cup \mathsf{Gdr}$ can be applied in parallel to G if both $\varphi \circ \psi$ and $\psi \circ \varphi$ are applicable to G.*

The following result is straightforward to prove and provides a simple criterium for two rules to be applicable in parallel.

Theorem 1. *Let $G = (V, E, \sigma)$ be a signed graph and let $\varphi, \psi \in \mathsf{Gnr} \cup \mathsf{Gpr} \cup \mathsf{Gdr}$ be two rules applicable to G with $\mathsf{dom}(\varphi) \cap \mathsf{dom}(\psi) = \emptyset$.*

(i) If $\varphi \in \mathsf{Gnr}$, then φ and ψ can be applied in parallel to G.
(ii) If $\varphi = \mathsf{gpr}_p$ with $p \in V$, then φ and ψ can be applied in parallel to G iff $N_G(p) \cap \mathsf{dom}(\psi) = \emptyset$.
(iii) If $\varphi, \psi \in \mathsf{Gdr}$, then φ and ψ can applied in parallel to G iff the subgraph of G induced by $\mathsf{dom}(\varphi) \cup \mathsf{dom}(\psi)$ is not isomorphic to C_4 or D_4.

Example 2. Let G be the signed overlap graph associated to the *actin I* gene in *S. nova*, shown in Fig. 4.
(i) The rules gnr_2, gnr_4, $\mathsf{gdr}_{5,6}$, and $\mathsf{gdr}_{8,9}$ can be applied in parallel to G.
(ii) The rules $\mathsf{gdr}_{5,6}$ and $\mathsf{gdr}_{6,7}$ are not applicable in parallel to G since applying one of them removes vertex 6 and thus, makes the other one inapplicable. Also, by Theorem 1, rules gpr_3 and $\mathsf{gdr}_{8,9}$ are not applicable in parallel to G.
(iii) There are only 6 different maximal parallel strategies to reduce this graph:

- $\{\mathsf{gnr}_2, \mathsf{gnr}_4, \mathsf{gdr}_{5,6}, \mathsf{gdr}_{8,9}\}\{\mathsf{gnr}_7, \mathsf{gpr}_3\}$;
- $\{\mathsf{gpr}_2, \mathsf{gnr}_4, \mathsf{gdr}_{6,7}, \mathsf{gdr}_{8,9}\}\{\mathsf{gnr}_5, \mathsf{gpr}_3\}$;
- $\{\mathsf{gpr}_2, \mathsf{gnr}_4, \mathsf{gdr}_{5,7}, \mathsf{gdr}_{8,9}\}\{\mathsf{gnr}_6, \mathsf{gpr}_3\}$;
- $\{\mathsf{gpr}_2, \mathsf{gpr}_3, \mathsf{gnr}_4, \mathsf{gdr}_{5,6}\}\{\mathsf{gnr}_7, \mathsf{gpr}_8, \mathsf{gpr}_9\}$;
- $\{\mathsf{gpr}_2, \mathsf{gpr}_3, \mathsf{gnr}_4, \mathsf{gdr}_{5,7}\}\{\mathsf{gnr}_6, \mathsf{gpr}_8, \mathsf{gpr}_9\}$;
- $\{\mathsf{gpr}_2, \mathsf{gpr}_3, \mathsf{gnr}_4, \mathsf{gdr}_{6,7}\}\{\mathsf{gnr}_5, \mathsf{gpr}_8, \mathsf{gpr}_9\}$.

Note that there are 3060 sequential strategies to reduce this graph (and assemble the gene) – the reason for this difference is that many sequential strategies coincide modulo commutation of some rules – as it turns out, these rules may be applied in parallel.

According to our definition, if a set of rules is applicable in parallel to a signed graph, then any composition of these rules is applicable to that graph. It is important to note that this definition only presumes that the rules are applicable in any possible order. However, this is enough to ensure that the result is always the same regardless of the order in which they are applied, as shown in the next two theorems.

We consider first the case of two rules and prove that if both $\varphi \circ \psi$ and $\psi \circ \varphi$ are applicable to a graph G, then $(\varphi \circ \psi)(G) = (\psi \circ \varphi)(G)$.

Theorem 2. *If $\varphi, \psi \in \mathsf{Gnr} \cup \mathsf{Gpr} \cup \mathsf{Gdr}$ are applicable in parallel to the signed graph G, then $\varphi(\psi(G)) = \psi(\varphi(G))$.*

Proof. If $\varphi \in \mathsf{Gnr}$ or $\psi \in \mathsf{Gnr}$, then the result is trivial. The rest of the cases follow observing that for any $S, S_1, S_2 \subseteq V$ and $p, q \in V$, $\mathsf{com}_S(G) - p = \mathsf{com}_S(G - p)$, $(G - q) - p = (G - p) - q$, and $\mathsf{com}_{S_1}(\mathsf{com}_{S_2}(G)) = \mathsf{com}_{S_2}(\mathsf{com}_{S_1}(G))$. Indeed, all our rules can be expressed as compositions of com and vertex removals. $\square$

The general case follows now easily from Theorem 2.

Theorem 3. *Let G be a signed graph and let $S \subseteq \mathsf{Gnr} \cup \mathsf{Gpr} \cup \mathsf{Gdr}$ be a set of rules applicable in parallel to G. Then for any two compositions φ, φ' of the rules in S, $\varphi(G) = \varphi'(G)$.*

Proof. There is a sequence $\varphi = \varphi_0, \varphi_1, \ldots, \varphi_m = \varphi'$ of permutations of φ, where

$$\varphi_i = \varphi_{i2}\alpha_i\beta_i\varphi_{i1} \quad \text{and} \quad \varphi_{i+1} = \varphi_{i2}\beta_i\alpha_i\varphi_{i1},$$

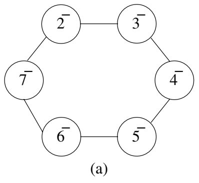
for some compositions φ_{i1} and φ_{i1} and rules α_i and β_i. Therefore, it is sufficient to show the claim for the case where the compositions are of the form $\varphi = \varphi_2\alpha\beta\varphi_1$ and $\varphi' = \varphi_2\beta\alpha\varphi_1$ for rules α and β. Also, in this case, $\varphi(G) = \varphi'(G)$ if and only if $\alpha\beta(\varphi_1(G)) = \beta\alpha(\varphi_1(G))$. Since $\alpha\beta$ and $\beta\alpha$ are both applicable to $\varphi_1(G)$, the claim follows from Theorem 2. $\square$

Clearly, if the rules in S are applicable in parallel to the signed graph G, then the rules in any subset of S are applicable in parallel to G. However, the reverse is not true, as shown by the following example.

Example 3. Let G be the all-negative hexagon in Fig. 7(a). Then any two rules from the set $S = \{\mathsf{gdr}_{2,3}, \mathsf{gdr}_{4,5}, \mathsf{gdr}_{6,7}\}$ are applicable in parallel to G by Theorem 1. However, the rules in S are not applicable in parallel to G. Indeed, applying any two of them to G in an arbitrary order makes the third one non-applicable. E.g., $(\mathsf{gdr}_{2,3} \circ \mathsf{gdr}_{4,5})(G)$ is the isolated all-negative graph on the set of vertices $\{6, 7\}$. Clearly, $\mathsf{gdr}_{6,7}$ is not applicable to this graph.

The following problem seems to be difficult: check whether or not an arbitrary given set of rules can be applied in parallel to a given signed graph. We give in the next theorem a simple criterium in the case where at most two rules gdr are among the rules.

Theorem 4. *Let G be a signed graph and $S \subseteq \mathsf{Gnr} \cup \mathsf{Gpr} \cup \mathsf{Gdr}$ a set of rules containing at most two rules gdr. Let P be the union of domains of rules in S with $P^+ = \{p \in P \mid \sigma(p) = +\}$ and $P^- = P \setminus P^+$. Then the rules in S can be applied in parallel to G if and only if the following conditions are satisfied:*



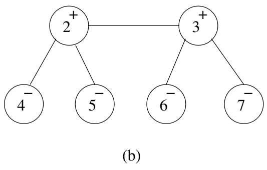


Fig. 7. (a) An all-negative graph irreducible in less than two steps; (b) A signed graph irreducible in less than four parallel steps

(i) The subgraph induced by P^+ is discrete. Moreover, there is no edge between vertices in P^+ and vertices in P^-.
(ii) The subgraph induced by P^- does not contain induced squares C_4 or diamonds D_4.

Note however that Theorem 4 does not hold when more than two rules gdr are in the considered set of rules as shown in Example 3. Indeed, for the graph in Fig. 7(a), $\mathsf{gdr}_{2,3}, \mathsf{gdr}_{4,5}, \mathsf{gdr}_{6,7}$ are not applicable in parallel, although no four vertices from $\{2, 3, \ldots, 7\}$ induce a subgraph isomorphic to C_4 or D_4.

4 Parallel Complexity of Micronuclear Genes

A new natural notion of complexity can be defined for the process of gene assembly using the notion of parallelism. We investigate this notion in the following and show how it leads to several intriguing questions on signed graphs.

Definition 2. *The* parallel complexity *of a micronuclear gene (and of its signed graph) is the minimal number of steps needed to reduce in parallel the signed graph associated to that gene.*

Example 4. (i) Consider the micronuclear gene *C2* in *S. nova* – its associated signed graph is the all-negative discrete graph with four vertices. Thus, its parallel complexity is 1.
(ii) Consider the micronuclear gene *actin I* in *S. nova* with the associated signed graph in Fig. 4. Its parallel complexity is two and a parallel strategy in two steps is $\{\mathsf{gnr}_2, \mathsf{gnr}_4, \mathsf{gpr}_3, \mathsf{gdr}_{5,6}\}$ $\{\mathsf{gnr}_7, \mathsf{gpr}_8, \mathsf{gpr}_9\}$, see also Example 2.
(iii) The signed graph G associated to the micronuclear gene αTP in *S. nova* consists of one negative clique G_1 with vertices $\{2, 3, \ldots, 12\}$ and one negative discrete subgraph G_2 with vertices $\{13, 14\}$. Its parallel complexity is two and a two step parallel strategy is $\{\mathsf{gpr}_2, \mathsf{gnr}_{13}, \mathsf{gnr}_{14}\}$ $\{\mathsf{gnr}_3, \mathsf{gnr}_4, \ldots, \mathsf{gnr}_{12}\}$.
(iv) The signed graph in Fig. 7(a) has parallel complexity two and a parallel strategy in two steps is $\{\mathsf{gdr}_{2,3}, \mathsf{gdr}_{4,5}\}\{\mathsf{gnr}_6, \mathsf{gnr}_7\}$.
(v) The signed graph in Fig. 7(b) has parallel complexity four and a parallel strategy in four steps is $\{\mathsf{gpr}_2\}\{\mathsf{gdr}_{3,6}\}\{\mathsf{gpr}_4, \mathsf{gnr}_7\}\{\mathsf{gnr}_5\}$.

Some upper and lower bounds on the parallel complexity of a signed graph are given in the next result.

Lemma 1. *(i) The parallel complexity of a signed graph with n vertices is at most $n/2 + 4$. If the graph is all-negative, then its complexity is at most $n/4 + 2$.*

(ii) There are signed graphs with parallel complexity four. There are all-negative graphs with parallel complexity two.

As it turns out, it is difficult to find signed graphs with parallel complexity higher than 5, or all-negative graphs with parallel complexity higher than 2. As a matter of fact, we conjecture in Section 5 that all-negative graphs have parallel complexity at most two! There are at least two types of graphs that seem intuitively "difficult to reduce" and could thus be good candidates for higher complexity: graphs for which no two rules can be applied in parallel in the first step, and graphs that avoid a certain rule (such as gdr that reduces two vertices at once) in all parallel reductions. We characterize these two types of graphs in the following.

Graphs with no parallelism in the first step

An induced subgraph H is said to be a *shadow* of a vertex set (or a subgraph) A if for each $x \in A$ and edge uv of H, x is adjacent to u or v or both, and each isolated vertex of H is adjacent to a vertex in A.

Theorem 5. *Let G be a signed graph of at least two vertices. Then G has no parallel applications of the rules (*gnr, gpr, *and* gdr*) if and only if*

(i) G^+ is a clique,
(ii) G^- is a shadow of G^+, and
(iii) $G^- = D \oplus T$, where D is a discrete graph and T is a complete tripartite graph.

Graphs that avoid one type of reduction

The signed graphs that have no reductions using gnr are those that can be reduced using only gpr and gdr. A string-based characterization was given in [3], but giving a similar graph-based characterization remains an open problem.

Theorem 6. *A signed graph G has no reductions using* gpr *if and only if G is all-negative.*

Theorem 7. *Let G be a connected signed graph with no reduction using* gdr*. Then $G = G^+ \otimes G^-$, G^+ is either a clique, or a disjoint union of two cliques, and G^- is discrete. Moreover, G can be reduced in at most three parallel steps.*

5 Conclusions

We have investigated in this paper a notion of parallelism for reducing signed graphs such as those associated to micronuclear gene patterns. We also introduced a notion of parallel complexity for micronuclear genes and their signed graphs, given by the minimal number of steps needed in a parallel reduction. Surprisingly, we have been unable to find examples of graphs with high parallel complexity; we conjecture that no such graphs exist. More specifically, we state the following conjectures:

Conjecture 1. All-negative graphs can be reduced in parallel in at most two steps.

There are many other interesting questions related to parallelism and signed graphs. E.g., although signed graphs require in general more parallel steps to reduce than all-negative graphs, we have been unable to find signed graphs with parallel complexity higher than 4. It remains an open question whether the parallel complexity of signed graphs is indeed bounded.

Acknowledgements

The authors are supported by European Union project MolCoNet, IST-2001-32008. T. Harju gratefully acknowledges support by Academy of Finland, project 39802. I. Petre gratefully acknowledges support by Academy of Finland, project 203667. G. Rozenberg gratefully acknowledges support by NSF grant 0121422.

References

1. Ehrenfeucht, A., Harju, T., Petre, I., Prescott, D. M., and Rozenberg, G., Formal systems for gene assembly in ciliates. *Theoret. Comput. Sci.* **292** (2003) 199–219.
2. Ehrenfeucht, A., Harju, T., Petre, I., and Rozenberg, G., Patterns of micronuclear genes in cliates. *Lecture Notes in Comput. Sci.* **2340** (2002) 279–289.
3. Ehrenfeucht, A., Harju, T., Petre, I., and Rozenberg, G., Characterizing the micronuclear gene patterns in ciliates. *Theory of Comput. Syst.* **35** (2002) 501–519.
4. Ehrenfeucht, A., Harju, T., Petre, I., Prescott, D. M., and Rozenberg, G., *Computation in Living Cells: Gene Assembly in Ciliates*, Springer (2003).
5. Ehrenfeucht, A., Petre, I., Prescott, D. M., and Rozenberg, G., Universal and simple operations for gene assembly in ciliates. In: V. Mitrana and C. Martin-Vide (eds.) *Words, Sequences, Languages: Where Computer Science, Biology and Linguistics Meet*, Kluwer Academic, Dortrecht (2001) pp. 329–342.
6. Ehrenfeucht, A., Petre, I., Prescott, D. M., and Rozenberg, G., String and graph reduction systems for gene assembly in ciliates. *Math. Structures Comput. Sci.* **12** (2001) 113–134.
7. Ehrenfeucht, A., Petre, I., Prescott, D. M., and Rozenberg, G., Circularity and other invariants of gene assembly in cliates. In: M. Ito, Gh. Păun and S. Yu (eds.) *Words, semigroups, and transductions*, World Scientific, Singapore (2001) 81–97.
8. Ehrenfeucht, A., Prescott, D. M., and Rozenberg, G., Computational aspects of gene (un)scrambling in ciliates. In: L. F. Landweber, E. Winfree (eds.) *Evolution as Computation*, Springer, Berlin, Heidelberg, New York (2001) pp. 216–256.
9. Jahn, C. L., and Klobutcher, L. A., Genome remodeilng in ciliated protozoa. *Ann. Rev. Microbiol.* **56** (2000), 489–520.
10. Prescott, D. M., The evolutionary scrambling and developmental unscabling of germlike genes in hypotrichous ciliates. *Nucl. Acids Res.* **27** (1999), 1243 – 1250.

11. Prescott, D. M., Genome gymnastics: unique modes of DNA evolution and processing in ciliates. *Nat. Rev. Genet.* 1(3) (2000) 191–198.
12. Prescott, D. M., Ehrenfeucht, A., and Rozenberg, G., Molecular operations for DNA processing in hypotrichous ciliates. *Europ. J. Protistology* **37** (2001) 241–260.
13. West, D. B., *Introduction to Graph Theory*, Prentice Hall, Upper Saddle River, NJ (1996).

Splicing Systems for Universal Turing Machines

Tero Harju[1] and Maurice Margenstern[2]

[1] University of Turku, Department of Mathematics,
20014 Turku, Finland
harju@utu.fi
[2] LITA, EA3097, UFR MIM, Université de Metz,
Île du Saulcy, 57045 Metz, France
margens@sciences.univ-metz.fr

Abstract. In this paper, we look at extended splicing systems (i.e., H systems) in order to find how small such a system can be in order to generate a recursively enumerable language.

It turns out that starting from a Turing machine M with alphabet A and finite set of states Q which generates a given recursively enumerable language L, we need around $2\times|I|+2$ rules in order to define an extended H system $\mathcal{H}$ which generates L, where I is the set of instructions of Turing machine M. Next, coding the states of Q and the non-terminal symbols of $\mathcal{L}$, we obtain an extended H system $\mathcal{H}_1$ which generates L using $|A|+2$ symbols. At last, by encoding the alphabet, we obtain a splicing system $\mathcal{U}$ which generates a universal recursively enumerable set using only two letters.

Keywords: DNA computing, splicing systems.

1 Introduction

Splicing systems are one of the broadest concepts of DNA computing, and so many papers deal with various aspects of what is possible to do with splicing systems that we cannot quote all of them, see [11].

Let us say simply that in most papers, the construction of the language which is computed by the splicing system is the same as considered in [11]. For this approach, let L be a language and denote by $\widetilde{\sigma}(L)$ the result of the application of the rules of a splicing system $\mathcal{S}$ to L. The language which is produced by $\mathcal{S}$ is given by $\bigcup\limits_{i\in\mathbb{N}} L_i$, where $L_0 = A$, the set of axioms, and language L_{i+1} is defined by: $(*)$ $L_{i+1} = \widetilde{\sigma}^{i+1}(L) = \widetilde{\sigma}^i(L)\cup\widetilde{\sigma}\widetilde{\sigma}^i(L)$. This is the case, for instance, in [3, 4, 2], where the question is also approached through multiplicities. In such splicing systems, each generation contains all the information of the previous generations.

In [5], we defined another approach: we *do not* assume that the elements of a generation survive to the next generation. Our approach can be formalised by the following scheme:
$(**)$ $\sigma^{i+1}(L) = \sigma\sigma^i(L)$.

We consider finite sets of axioms and finite sets of rules. For the processing operation $(*)$, it was proved by Culik and Harju, [1] (see also Pixton, [12]) that the generated splicing system is regular. For the nonpreserving operation $(**)$, we show that all recursively enumerable languages can be generated.

This result means that the nonpreserving operation introduce some control on the process which explains the possibility of universal computations. This results is to be compared with other results on extensions of splicing systems where various means of control are introduced, in particular, by elimination of molecules which cannot enter a rule, see for instance [8, 9, 15] with time-varying distributed systems and especially for [9] which is the closest to our definition but not exactly the same.

In the first section, we remind the definitions about splicing systems and we give the definition of our approach.

In the second section, we remind our universality results of [5] and we give a method in order to obtain a rather **small** splicing systems which can generate any recursively enumerable language.

2 Splicing Systems and Turing Machine Simulations

A splicing system $\mathcal{S}$ is a triple (Σ, A, R), where Σ is a finite alphabet, A is a finite set of words called **axioms** and R is a finite set of **rules** which we presently define.

A rule of $\mathcal{S}$ is given by four words in Σ^*, say (u_1, u_2u_3, u_4) which we shall display as follows:

$$\begin{array}{c|c} u_1 & u_2 \\ \hline u_3 & u_4 \end{array}$$

In the literature, the same rule is also often displayed as $u_1\#u_2\$u_3\#u_4$.

The application of a rule to a pair of words (w_1, w_2) can be defined as follows:

- if w_1 contains an occurrence of u_1u_2, say $w_1 = x_1u_1u_2y_1$, and w_2 contains an occurrence of u_3u_4, say $w_2 = x_2u_3u_4y_2$, then rule $u_1\#u_2\$u_3\#u_4$ applies to (w_1, w_2) and the result of the application is $(x_1u_1u_4y_2, x_2u_3u_2y_1)$.

  This application can be denoted by:

$$\begin{array}{c|c} x_1u_1 & u_2y_1 \\ \hline x_2u_3 & u_4y_2 \end{array} \vdash_r \left\{ \begin{array}{l} x_1u_1u_4y_2 \\ x_2u_3u_2y_1 \end{array} \right.$$

- if w_1 does not contain u_1u_2 or if w_2 does not contain u_3u_4, then we say that rule $u_1\#u_2\$u_3\#u_4$ does not apply to (w_1, w_2).

When rule r applies to (w_1, w_2) with (y, z) as a result, we denote this by $(w_1, w_2) \vdash_r (y, z)$.

The **language generated** by $\mathcal{S}$ which we denote by $\mathcal{L}(\mathcal{S})$ is defined as follows:

$$\mathcal{L}(\mathcal{S}) = \bigcup_{n \in I\!N} \sigma^n(A)$$

where $\sigma^0(A) = A$ and
$\sigma(\mathcal{M}) = \{w \; ; \; \exists w_1, w_2, z \in \mathcal{M}, \exists r \in R(w_1, w_2) \vdash_r (w, z) \;\text{ or }\; (w_1, w_2) \vdash_r (z, w) \;\}$,
with $\mathcal{M}$ running over $\sigma^n(A)$, compare with $(**)$.

We also consider **extended** splicing systems.

They are obtained by changing the alphabet and the definition of the generated language in the following way. Now, we consider that $\Sigma = T \cup N$, where T is called **terminal alphabet** and N is called the **set of non-terminal symbols** and the system itself, say $\mathcal{E}$, is denoted by (T, N, A, R). Also, the language generated by $\mathcal{E}$ is defined by $\mathcal{L}(\mathcal{E}) = \mathcal{L}(\mathcal{S}) \cap T^*$, where $\mathcal{S} = (T \cup N, A, R)$.

As indicated with full details in [5], extended splicing systems can simulate deterministic Turing machines with a single head and a single bi-infinite tape.

The idea of the simulation is that going from a current configuration to the next one in the computation of Turing machine is a **local** transformation of the current configuration. As splicing is also local to the sites where the rule operates, we may expect to represent one step of the computation of a Turing machine by the application of a splicing rule.

This is the case and our report [5] gives an explicit set of rules for that purpose.

We have just to mention a point which is connected with the simulation within **finite** strings of a Turing configuration which is a finite part of the **infinite** tape.

This means that the treatment of the ends of a word is different for Turing machines and splicing systems. We solved this problem by introducing a special marker which indicates the ends of the Turing configuration. When the signal which simulates the Turing machine head meats the marker, it removes it by one square further, putting in its place a blank. This is not difficult to implement in splicing rules, see [5].

In [5], we proved the following result:

Theorem 1. *For any RE language $\mathcal{M}$, there is an extended splicing system $\mathcal{E}$ such that $\mathcal{L}(\mathcal{E}) = \mathcal{M}$.*

Using the schemes for splicing rules introduced in [5], we could prove that:

Corollary 1. *Let $\mathcal{M}$ be an RE language on $\{a, b\}$ which is simulated by a Turing machine M with k instructions. There is a splicing system having $2k + 32$ rules which generates $\mathcal{M}$.*

Also, using a coding of the alphabet of the recursively enumerable set by only two letters, we obtained in [5]:

Corollary 2. *For each RE language $\mathcal{M} \subseteq \Gamma^*$, there is a non-extended splicing system $\mathcal{S}$ and a coding $c : \; \Gamma^* \mapsto \{0, 1\}^*$ such that*

$$\mathcal{M} = c^{-1}\Big(L(\mathcal{S})\Big)$$

3 Universality Results

From theorem 1 we know that for each RE language, we can construct an extended splicing system which generates it.

In [5], we proved that there is a uniform way to do this by using universal Turing machines:

Theorem 2. *There is an extended splicing system $\mathcal{E} = (T, N, A, R)$ and an encoding c over T^* such that for any RE $\mathcal{M}$, there is a word $w_{\mathcal{M}}$ such that the new system $\mathcal{E}' = (T, N, A \cup \{w_{\mathcal{M}}\}, R)$ generates $c^{-1}(\mathcal{M})$.*

The idea of the proof is to simulate a universal Turing machine U and $w_{\mathcal{M}}$ is a suitable encoding of a Turing machine M which generates $\mathcal{M}$.

We shall follow the same idea in a somehow more sophisticated pattern in order to find an extended system which generates recursively enumerable languages with a **small** number of rules.

In the late fifties and early sixties of the previous century, there was a race to find the smallest universal Turing machines. A long pause was put on this race by the results of Yurii Rogozhin who, in 1982, devised seven very small universal Turing machines, see [13, 14]. From these machines, we take the one which has seven states and four symbols. It is usually denoted by $UMT(7,4)$.

These machines simulate tag systems which are proved to be universal, see [10]. Tag-systems are defined as follows. We have an alphabet A, a positive number p and a mapping $a_i \mapsto P_i$ from A into A^*, P_i's being called the **productions**.

One step of computation is defined by the following process where w is the **current word**:

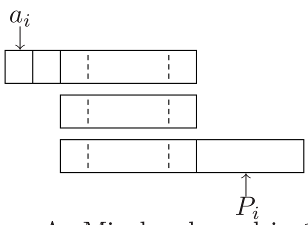
let a_i be the first letter of w;
erase first p letters of w, and let w' be what remains;
append P_i to w'.

The computation starts again with $w'P_i$ as the new current word. It halts by meeting a **halting** letter in first position. We may assume that there is a single halting letter and we denote it by **!**.

As an example, consider tag-system P on {a,b,c} with current word bbb:

P:	applied to bbb:
a $\longrightarrow$ b	bb|b
b $\longrightarrow$ bc	|bb|c
c $\longrightarrow$!	| |c !

The general scheme for $p = 2$ can be simulated as follows:



As Minsky showed in 1962 that tag-systems can simulate Turing machines, and as tag-systems can be easily simulated by Turing machines, this opened the way to very small universal Turing machines. Below, the first seven lines of Table 1, except the second instruction, are taken from Rogozhin's $UTM(7,4)$.

Table 1. Program of machine U'

	$\mathbf{0}$	$\mathbf{1}$	b	c
1	L	$\mathbf{0}\,L$	$c\,R\,2$	$b\,L$
2	$\mathbf{1}\,R$	$\mathbf{0}\,L\,1$	$c\,R$	$\mathbf{1}\,R\,5$
3	$\mathbf{1}\,L\,4$	R	$c\,R$	$b\,R$
4	$\mathbf{1}L\,7$	L	$c\,L$	$b\,L$
5	$c\,L\,4$	R	$c\,R$	$b\,R$
6	$R\,5$	$\mathbf{0}\,R$	R	$\mathbf{0}\,R\,1$
7	$R\,3$	$R\,8$	$L\,6$	_
8		$\mathbf{0}\,R$	R	$\mathbf{0}\,R\,9$
9	$c\,R\,\mathbf{!}$	R		R

Without entering in the technique of the simulation, in a first stage, the machine locates the production to be appended to the current word. To do so, the encoding of the letters in the current word contain as many symbols as there are markers in the encoding of the productions between the letter and its production. To give a better idea to the reader, the tape of $UTM(7,4)$ looks like this:

$$\mathbf{10}P_n \dots P_i \dots P_1 P_0 L_1 c L_2 c \dots c L_k$$

where $L_i = \mathbf{1}^{N_i}$ with N_i being the number of b's to be marked between L_i and its corresponding production, and where P_i is a concatenation, in reverse order, of codes of the letters in the form $b\mathbf{00}^{N_i}$, and P_i itself starts with an additional b.

When production P_i, corresponding to L_1 is located, the tape looks like this:

$$\mathbf{10}P_n \dots P_i\boxed{\dots P_1 P_0 L_1}\underset{\triangle}{c}L_2 c \dots c L_k$$

where, inside the frame, b's are replaced by c's and 0's by 1's.

We are interested in the aspect of the tape when the halting letter is the first one. In that case, the tape looks like this:

$$\mathbf{10}\boxed{P_n \dots P_i \dots P_1 P_0 L_1}\underset{\triangle}{c}L_2 c \dots c L_k$$

with the same transformation of the tape as previously in the part of the tape within the frame. The triangle indicate the position of the head which scans a c under state 2. Next, the head goes on to the right under state 5 until it meets the leftmost **0** which it replaces by a c. Then, the head goes back to the left under state 4, leaving **1**'s unchanged and transforming b's into c's and then c's into b's, until the head meets the rightmost **0**. It replaces **0** by **1** and goes to the left under state 7. There it meets a **1**, which means that the computation of the tag system is completed. Table 1 appends instructions to Rogozhin's $UTM(7,4)$, giving us a new machine U', in order to restore the encoding of the tag system and to prepare the collection of a word which belongs to the resulting language. This is why crossing back the configuration, the head arrives to its right-hand end where it halts on the leftmost **0**.

In terms of extended splicing systems, this means that we arrive to a word whose right-hand end is of the form $q_f\#$. Rules (B_1), (B_2), (s_1) and (s_2) allows us to remove # from the right-hand end of the word and to go leftwards until the rightmost **0** is met: what is on the right hand of this letter is the word to append to the language. This is performed by rules similar to rule (C_1) with $*$ replaced by **0**. We may notice here that as the result of the tag-system has always at least three symbols, we do not need rule (C_2).

At this point, the rôle of rule (I_1) is replaced by a much more complicate process. First, we have to destroy all **0**'s which we meet, still going to the left, until the rightmost occurrence of b is reached. We append a new symbol, d, to the right hand of b and we start the program of a new Turing machine V. Let us call $\mathcal{T}$ the encoding of the tag-system on the tape: its right-hand end is d and its left-hand end is **1** as far as in between, there are only **0**'s and b's. The rôle of V is to put the encoding of the next initial current word on the right hand of $\mathcal{T}$.

Recall that the initial current word corresponds to the encoding of configuration (3). Recall that the tag system to which we apply U' does not directly simulate Turing machine K in the proof of theorem 1. It simulates it through a register machine R_1 with two registers which simulates a register machine R_0 with three registers simulating K. The simulation of R_0 by R_1 entails an exponential slowdown: if the registers of R_0 contain non-negative integers x, y and z, the registers of R_1 contain $2^x3^y5^z$ and 0 at an appropriate step. As the initial configuration of R_0 is $x, 0, 0$, the initial configuration of R_1 is $2^x, 0$. Now, if we encode configuration (3) which is essentially n in unary, we get that $x = 2^n$. In order to avoid a double exponential, we encode n in binary. Accordingly, machine K must be replaced by a machine K' with a two letter alphabet which does the same as K and in which n is encoded in binary: **0**, **1** and $*$ of the tape of the new machine K are respectively encoded as **10**, **11** and **01** and we reserve **00** to encode the blank of K. As **10** cannot be confused with the blank of the tape of machine K, we use $*$ as a separator. Call $*$ the blank cell which is the left neighbour of the leftmost **1** on the tape of K'. We also can assume that K' does not go to the left of *ast*. It is enough to guarantee this for the successor in the proof of Kleene's theorem. As there is no difficulty, we leave the easy details to the reader.

Accordingly, we may assume that $x = n$. The next value of x will be $n+1$. Let $u = 2^x$. This means that the new value of u is $2^{n+1} = 2.2^n$.

Turning now to the tag-system simulation, u, v, the contents of the registers of R_1 are encoded as $Aa(aa)^uBb(bb)^v$. We may assume that the current word which corresponds to configuration (3) is $Aa(aa)^uBbbb$ with $u = 2^n$, where A, a, B and b are fixed letters of the alphabet of the tag system. We may assume that a, A, b and B are respectively encoded by $\mathbf{1}c$, $\mathbf{111}c$, $\mathbf{11}c$ and $\mathbf{1111}c$.

Summarising all this information, when machine U' halts, the initial current word of the tag-system goes from

$$\mathbf{111}c\mathbf{1}c(\mathbf{1}c\mathbf{1}c)^u\mathbf{1111}c\ \mathbf{11}c\mathbf{11}c\mathbf{11}c$$
$$\mathbf{111}c\mathbf{1}c(\mathbf{1}c\mathbf{1}c)^{2u}\mathbf{1111}c\ \mathbf{11}c\mathbf{11}c\mathbf{11}c.$$

For that purpose, we keep a copy of $(\mathbf{1}c)^u(d\mathbf{1})^{10}$ which we put on the left hand of the encoding $\mathcal{T}$ of the tag-system on the tape. We introduce an additional letter, d, which we use in such a way that during the program of multiplication by 2, we use motions between two $\mathbf{0}$'s. Letter d is also used to encode a fixed part of the initial current word which corresponds to the encoding of A and of $Bbbb$.

The installation of the next initial current word is performed by machine V whose table is displayed by Table 2.

Machine V marks with d the rightmost $\mathbf{0}$ of $\mathcal{T}$ which also indicates its right-hand end: state 1. Then, it goes to the left-hand end of the configuration, marking by $\mathbf{1}$ the $\mathbf{0}$'s of T: state 2; the motion goes on with state 3 until the leftmost $\mathbf{0}$ is reached. State 4 is a test for loop (L_1) which erases $(\mathbf{1}c)^u$ while copying it onto $(\mathbf{1}c)^{2u}$. On state 4, the head of V erases a $\mathbf{1}$ to meet a c or a d. If it meets a d, it is the end of (L_1). If it is a c, a new round of the copying action is to be performed. State 5 puts the head to the other end of the configuration, on the leftmost $\mathbf{0}$. There, it writes down $(\mathbf{1}c)^2$, using the instructions of states 21 up to 23 corresponding to $\mathbf{0}$. State 23 sends the head to the left. The motion is controlled by state 6 which halts it on the rightmost $\mathbf{0}$ which marked a c. When $\mathbf{0}$ is reached, the head moves by one step to the right and a new test occurs.

When loop (L_1) is completed, a new loop, (L_2) occurs: it copies $(d\mathbf{1})^{10}$ on the left hand of $\mathcal{T}$ to $(\mathbf{1}c)^{10}$ at the right-hand end of the configuration. The test of the loop is performed by state 9, the motion to the right is controlled by state 7, the motion to the left by state 8. The writing on $\mathbf{0}$ is made by states 7 and 28. The latter state calls state 8 which transfers the control to the test of state 9 when the head has performed its motion to the left. The test looks whether it reads d, in which case a new cycle of copying. If it reads b, the head knows that this second round of copying is completed. State 11 puts the head on the right-hand end of the tape and from there, the instructions of states 21 up to 27 scan $\mathbf{1}$'s and c's of this end of the tape and they change it such that $Bbbb$ appears at the end of the encoding. When this is done, state 27 marks with d the first letter of the encoding of $Bbbb$ which is a $\mathbf{1}$. Next, state 28 brings back the head on mark d of the right-hand end of $\mathcal{T}$: the head corrects the tape in such a way that its beginning encodes A, which is performed by state 29 and with the help of states 31 and 32, the head marks by d the last letter of the encoding of Aa which is a c. Then, the head goes on to right in a cycle of copying $\mathbf{1}c$ on the left hand of $\mathcal{T}$ for the computation of the following initial current word. This is performed by states 41 up to 44 where state 41 realises the test which controls the loop. The loop is completed when it meets d under state 41. It then restores the **1** which was marked by d and the machine goes to the left to the other d which marks the rightmost c of Aai: state 52. Then, it restores the **0**'s which are with b in $\mathcal{T}$.

The occurrence of d in state 53 indicates that the head is on the left of $\mathcal{T}$ and that it has wrongly changed a **1** into **0**. This is corrected by state 54 which also changes into **1** the leftmost letter of $\mathcal{T}$. The, state 55 brings the head to d where it restores **0** which should be there and it stops. We can now give the control to machine U'.

Table 2 could be made more compact but it would be much more difficult to understand it.

Table 2. Program of machine V

	0	**1**	b	c	d
1	$d\,L\,2$				
2	$\mathbf{1}\,L$	L	L		$L\,3$
3	$R\,4$	L		L	L
4		$\mathbf{0}\,R$		$\mathbf{0}\,R\,5$	$\mathbf{0}\,R\,7$
5	$\mathbf{1}R\,21$	R	R	R	R
6	$R\,4$	L	L	L	L
7	$\mathbf{1}R\,28$	R	R	R	R
8	$d\,R\,9$	L	L	L	L
9		R	$R\,11$		$\mathbf{0}\,R\,7$
11	$L\,21$	R	R	R	R
21	$c\,R\,22$	$L\,22$		L	
22	$1\,R\,23$	$c\,L\,23$		$\mathbf{1}\,L$	
23	$c\,L\,6$	$L\,24$		$\mathbf{1}\,L$	
24		$L\,25$		L	
25		$c\,L\,26$		$\mathbf{1}\,L$	
26		$L\,27$		$\mathbf{1}\,L$	
27		$d\,L\,28$		$\mathbf{1}\,L$	
28	$c\,L\,8$	L		L	$R\,29$
29		R		$\mathbf{1}\,L\,10$	
31		R		$R\,32$	
32		R		$dR\,41$	
41		R		$\mathbf{0}R\,42$	$\mathbf{1}\,L\,51$
42	$c\,L\,43$	L	L	L	L
43	$\mathbf{1}R\,44$				
44	$c\,R\,41$	R	R	R	R
51		L		L	$c\,L\,52$
52		L		L	$c\,L\,53$
53		$\mathbf{0}\,L$	L		$R\,54$
54	$\mathbf{1}\,R$		R		$\mathbf{0}\,R\,\mathbf{!}$

Now, let us count he number of rules which are needed.

Turing machine U' has 33 instructions, the halting one included. Machine V has 93 instructions, halting also included. This makes 126 instructions, hence, 252 rules. To detach the word which is computed by the tag system, we need rules

(B_1), (B_2), (s_1), (s_2) and an adaptation of rule (C_1) to this setting. As (B_2), (s_1), (s_2) and (C_1) are schemes of rules and as the alphabet has 5 letters, this gives us 21 rules.

To removing **0**'s after detaching the word which is produced, we need rules similar to (B_1) and $B(2)$ which means again 6 rules. We need an additional rule similar to (I_1) to transfer the control to V.

Accordingly, we need 28 rules for tasks on the simulated Turing tape which are typical for splicing systems and which cannot be performed by Turing machines. In total, we need 280 rules.

This gives us the following result:

Corollary 3. *There is an extended splicing system* $\mathcal{E} = (T, N, A, R)$ *such that* $T = \{\mathbf{0},\mathbf{1}\}$*, such that* R *contains* 280 *rules an such that there is an encoding* c *over* T^* *such that for any RE* $\mathcal{M}$*, there is a word* $w_{\mathcal{M}}$ *such that the new system* $\mathcal{E}' = (T, N, A \cup \{w_{\mathcal{M}}\}, R)$ *generates* $c^{-1}(\mathcal{M})$*.*

4 Conclusion

The last result of the paper points at how low the descriptional complexity of a universal splicing system can be. Using the well known Turing simulation of tag systems which allowed to obtain very small universal Turing machines, here, we obtained a rather small splicing system which is able to generate any recursively enumerable set. We did not completely investigate this aspect. It could also be of interest to find out how complex are the rules which are involved in corollary 3.

Also, notice that our result is highly *sequential* in its spirit, despite the highly parallel potentiality of the model. We think that this is in connection with our introductory remark about the sensibility of the computational of splicing systems with respect to the definition of the generation of the language. The well known regularity result of extended splicing systems with finitely many axioms and finitely many rules is also probably connected with the fact that the lack of control prevents the realisation of highly sequential processes. This relation between sequentialisation and universality was already noticed in [8, 9, 15]. However, these papers dealt with several splicing systems and the discussion was more on the communications between the systems. With the result on universality which holds also for non-extended splicing systems, we think that the present paper throws a new light on this connection.

Acknowledgement

Both authors acknowledge the help of IST-2001-32008 **MolCoNet** *project* for giving the best conditions to their cooperation.

References

1. Culik, K., II and Harju, T., Splicing semigroups of dominoes and DNA, *Discrete Appl. Math.*, **31**, (1991), 261–277.
2. Ferretti C. and Frisco P., Mauri G., Simulating Turing machines through extended *mH* systems, *Computing with Bio-Molecules. Theory and Experiments*, (G. Păun, ed.), Springer Verlag, (1998), 221–238.
3. Freund R., Kari L., and Păun G., DNA computation based on splicing: the existence of a universal computer, *Theory of Computing Systems*, **32**, (1999), 69–112.
4. Ferretti, C., Mauri, G., Kobayashi, S., and Yokomori, T. , On the universality of Post and splicing systems. Universal machines and computations (Metz, 1998). *Theoret. Comput. Sci.*, **231**, (2000), 157–170.
5. Harju, T. and Margenstern, M. Remarks on the universality of splicing systems, TUCS Report (2004).
6. Kleene S.C., A note on recursive functions, *Bulletin of the American Mathematical Society*, **42**, 544bis–546, (1936).
7. Margenstern M., Turing machines: universality and limits of computational power, in *Formal Languages and Applications*, C. Martin-Vide, V. Mitrana, Gh. Păun, ed., Springer-Verlag, to appear.
8. Margenstern M., Rogozhin Yu. About Time-Varying Distributed *H*-systems, *Lecture Notes in Computer Science*, **2054**, (2000), 53–62.
9. Margenstern M., Rogozhin Yu., Verlan S., Time-varying distributed H sytems with parallel computations: the problem is solved, *Lecture Notes of Computer Sciences*, **2943**, (2004), 48–53.
10. Minsky M.L. , Size and structure of universal Turing machines using Tag systems, *Proc. Sympos. Pure Math.*, **5**, (1962), 229–238.
11. Păun, Gh., Rozenberg, G., Salomaa, A., *DNA Computing*, Springer-Verlag, Berlin Heidelberg, 1998.
12. Pixton, D., Regularity of splicing languages, *Discrete Appl. Math.* **69**, (1996), 101–124.
13. Rogozhin Yu., Seven universal Turing machines, *Mat. Issled.*, No. 69, (1982), 76–90.
14. Rogozhin Yu., Small universal Turing machines, *Theoret. Comput. Sci.*, **168**, (1996), 215–240.
15. Verlan S., A frontier result on enhanced time-varying distributed H systems with parallel computations, *Preproceedings of DCFS'03, Descriptional Complexity of Formal Systems*, Budapest, Hungary, July 12-14, (2003), 221–232.

Application of Mismatch Detection Methods in DNA Computing

Christiaan V. Henkel[1,2], Grzegorz Rozenberg[2], and Herman P. Spaink[1]

[1] Institute of Biology, Leiden University,
Wassenaarseweg 64, 2333 AL Leiden, The Netherlands
{henkel, spaink}@rulbim.leidenuniv.nl
[2] Leiden Institute of Advanced Computer Science, Leiden University,
Niels Bohrweg 1, 2333 CA Leiden, The Netherlands
rozenber@liacs.nl

Abstract. In many implementations of DNA computing, reliable detection of hybridization is of prime importance. We have applied several well-established DNA mutation scanning methods to this problem. Since they have been developed for speed and accuracy, these technologies are very promising for DNA computing. We have benchmarked a heteroduplex migration assay and enzymatic detection of mismatches on a 4 variable instance of 3SAT, using a previously described blocking algorithm. The first method is promising, but yielded ambiguous results. On the other hand, we were able to distinguish all perfect from imperfect duplexes by means of a CEL I mismatch endonuclease assay.

1 Introduction

Biomolecular computing studies the potential of using biological molecules, currently mainly DNA, to perform computations. One focus of investigation is the use of combinatorial libraries of DNA to provide search spaces for parallel filtering algorithms. Many different methods for library generation, solution filtering, and output have been studied experimentally. The formal satisfiability problem has become a sort of benchmark in the implementation of such algorithms [1,2,3].

Computing by blocking is one of the methodologies for molecular computing [4]. The blocking algorithm uses nucleic acid complementarity to remove molecules not representing a solution from the candidate pool. To an initial library of single stranded DNA molecules corresponding to (all) potential solutions, a set of complementary falsifying DNA (blockers) is added. Only those library molecules not representing solutions will combine with a blocker to form a perfect DNA duplex. Library molecules corresponding to solutions should remain single stranded or form a duplex with mismatched basepairs, depending on experimental conditions. The experimental challenge in implementing this algorithm is to very precisely separate perfectly matched molecules from mismatched ones.

The original proposal for the implementation of the blocking algorithm was using PCR inhibition. Falsified molecules were to be made unavailable for DNA polymerase through their association with a blocker molecule, for example peptide nucleic acid (PNA). This would result in the selective amplification of unblocked DNA. So far, experimental data supporting this method is lacking. However, several fluorescence techniques (resonance energy transfer and correlation spectroscopy) have been successfully employed in combination with the blocking algorithm [5].

Here, we report the use of a heteroduplex migration assay and enzymatic mismatch recognition to implement blocking. In contrast to the fluorescence techniques mentioned before, both rely on DNA duplex structure rather than hybridization kinetics. These techniques are widely used to scan for mutations in molecular biological and clinical laboratories, and are well suited for high-throughput analysis of large numbers of samples [6,7].

During electrophoresis, perfect double stranded (homoduplex) DNA migrates through a gel at a predictable rate, dependent only on the strength of the applied electrical field, gel and buffer conditions and DNA length. However, DNA containing nucleotide mismatches (heteroduplex) and single stranded DNA migrate at anomalous rates, caused by secondary structure formation (ssDNA) or helix distortion (dsDNA). Such structures experience specific, but unpredictable, resistances when migrating through the gel matrix. Their mobility is lower than that of homoduplexes of equal length and as a result bands end up higher on the gel. Several well-established and sensitive mutation detection techniques exploit this effect, such as single strand conformational polymorphism (SSCP), temperature or denaturing gradient gel electrophoresis (TGGE, DGGE) and heteroduplex analysis [8].

Enzymatic mismatch recognition is also widely used in mutation detection [9]. It uses specific endonucleases which recognize and digest the abnormal DNA conformations which result from mismatched nucleotides. We have used the recently discovered CEL I nuclease, purified from celery, for this purpose [10,11].

2 Experimental

2.1 Problem Instance and Algorithm

We have tested mutation detection techniques on the following 4 variable, 4 clause satisfiability (3SAT) problem:

$$F = (\sim a \vee b \vee \sim c) \;\&\; (a \vee \sim b \vee d) \;\&\; (\sim a \vee c \vee \sim d) \;\&\; (b \vee c \vee \sim d)\,, \tag{1}$$

where a, b, c and d are the 4 variables with values of true (1) or false (0). $\vee$ stands for the OR operation, & for AND, ~ for negation. Since the clauses are connected by AND, falsifying one clause is sufficient for falsification of the whole formula. For example, falsification of the first clause by $abc = \{101\}$ falsifies the entire formula F.

The blocking algorithm proceeds as follows:

1. synthesize all possible assignments as ssDNA;
2. synthesize blockers representing to falsifying assignments;
3. mix and hybridize;
4. apply mismatch detection method.

Library/blocker combinations that form perfect duplexes correspond to false assignments.

2.2 Sequence Design

To represent the entire solution space to a 4 variable SAT problem, 16 library oligonucleotides were designed. The general structure of the library molecules is:

5' [*start*] [*a*] [*b*] [*c*] [*d*] [*stop*] ,

with *a*, *b*, *c* and *d* sequences representing variables. Two subsequences correspond to the two values these variables can take: ATCACC for *false*, and GTCTGA for *true*. The sequence of any variable thus only depends on its value, not on its identity. *start* and *stop* are invariable sequences (CTTGCA and TTGCAC, respectively), bringing the total length of the molecules to 36 nucleotides. Library molecules are numbered from 0 to 15, after the binary numbers they encode. For example, truth assignment *abcd* = {1010} is represented by oligonucleotide 10:

```
5’ CTTGCA GTCTGA ATCACC GTCTGA ATCACC TTGCAC
   [start]  [a = 1]  [b = 0]  [c = 1]  [d = 0]  [stop]
```

Falsifying oligonucleotides, or blockers, are complementary to the library oligonucleotides:

3' [*start*] [*a*] [*b*] [*c*] [*d*] [*stop*] ,

with *start* = GAACGA, *stop* = AACGTG, *true* = CAGACT and *false* = TAGTGG (all 3' $\rightarrow$ 5'). Since the falsification of a clause only requires 3 specified variables, and blocker molecules must contain a statement on all 4 variables, 2 blockers need to be designed for every clause. The fourth variable is set to true in one, and to false in the other. (It may be possible to circumvent this encoding complication through the use of redundant blockers, which contain universal nucleotides [12].) The translation of all clauses into blockers is summarized in table 1.

Table 1. Blocker molecules

Clause	Falsified by *abcd*	Blocker molecules
$(\sim a \vee b \vee \sim c)$	1010	A0
	1011	A1
$(a \vee \sim b \vee d)$	0100	B0
	0110	B1
$(\sim a \vee c \vee \sim d)$	1001	C0
	1101	C1
$(b \vee c \vee \sim d)$	0001	D0
	1001	identical to C0

Constraints on the design of the variable sequences were: GC content < 50%, isothermal melting behaviour (calculated according to [13,14]), no repeats or subsequence complementarity >2 bp, and no self complementarity. The uniform melting behaviour results in a T_m that is in theory identical for all possible library/blocker combinations.

2.3 Oligonucleotides

Oligonucleotides were custom synthesized and labelled at Isogen Bioscience (Maarssen, NL). For detection, library molecules contain a covalent 5' Cy5 label (Amersham Biosciences), blockers a 5' fluorescein (FITC, Molecular Probes). All oligos were purified from 10% denaturing polyacrylamide gels to remove unbound dye. DNA was allowed to diffuse from gel slices by overnight soaking in 0,5 M NH_4Ac, 2 mM EDTA, 0.1% SDS, and recovered by ethanol precipitation. Concentrations were calculated from absorption measurements of the dyes at 494 nm (fluorescein) or 649 nm (Cy5). Molar extinction coefficients of 77,000 cm^{-1} M^{-1} (fluorescein) and 250,000 cm^{-1} M^{-1} (Cy5) were used.

2.4 Duplex Migration Assay

Mixtures of library and blocker molecules were made by combining 5 pmol per oligo in a gel loading buffer consisting of 1x TBE, 3.3% sucrose and 0.033% Orange G. Duplex DNA was formed by heating the mixtures to 95 °C for 5 minutes, and cooling to 4 °C at 0.1 °C $second^{-1}$ in a thermocycler (Biometra TGradient). Gels were prepared from regular acrylamide: bisacrylamide (20:1) or proprietary SequaGel MD (Mutation Detection) acrylamide matrix (National Diagnostics, Atlanta, Georgia, USA). Duplex destabilizing chemicals (urea, ethylene glycol, formamide, or glycerol) were sometimes added to enhance heteroduplex migration effects [15]. Gels were run in 1x TBE at 200 V and 4 °C. Gel images were captures on a Biorad FluorS MultiImager, using UV excitation with 530 nm band pass and 610 nm long pass filters for detection of fluorescein and Cy5 fluorescence, respectively. Digital images were processed in Corel Photopaint 11 (contrast adjustment and greyscale conversion).

2.5 Enzymatic Mismatch Cleavage Assay

Duplexes were prepared as described above, except that hybridization was carried out in 10 mM Tris/HCl pH 8.5. T7 endonuclease I (T7EI) was obtained from New England Biolabs and handled according to the manufacturers recommendations. Reactions containing 5 pmol per oligonucleotide and 1 unit of enzyme were allowed to proceed for up to 150 minutes.

CEL I enzyme was obtained from Dr Edwin Cuppen (Hubrecht Laboratory, Utrecht, NL), see *http://cuppen.niob.knaw.nl* for a detailed isolation protocol. Several batches of varying activity were used throughout the experiments described in this article. Every lot of CEL I was tested, and for all subsequent experiments quantities were used that gave the effect shown in Fig. 3 after 30 minutes of incubation. Reactions were performed with 5 pmol per oligonucleotide in a 4 µl volume at 45 °C, in a 10 mM $MgSO_4$, 10 mM HEPES pH 7.5, 10 mM KCl, 0.002% Triton X-100, 0.2 µg $µl^{-1}$ BSA buffer. Reactions were stopped by placing samples on ice and adding 4 µl 80% formamide, 100 mM EDTA. Digests were analyzed on 10% TBE/polyacrylamide gels, which were imaged as before. Bands were analyzed using ImageJ software (version 1.31v, *http://rsb.info.nih.gov/ij*).

3 Results

3.1 Heteroduplex Migration

Optimal conditions for the heteroduplex migration assay were determined using several blocking and non-blocking oligo combinations and various gel formulations. 12.5% acrylamide gels supplemented with 20% urea were found to give good separation of duplexes and heteroduplexes and were used for all subsequent experiments (Fig. 1).

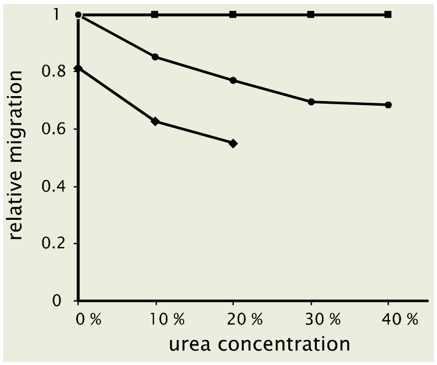


Fig. 1. Optimization of gel formulation. Shown are relative migration distances of 3 blocker/library combinations on 12.5% gels in the presence of progressive concentrations of urea. Migration of the perfect duplex combination C0 + 09 (squares) is set to 1. Other combinations are C0 + 08 (circles) and C0 + 10 (diamonds). The latter combination migrates as separate oligonucleotides on gels with urea concentrations of 30% and higher

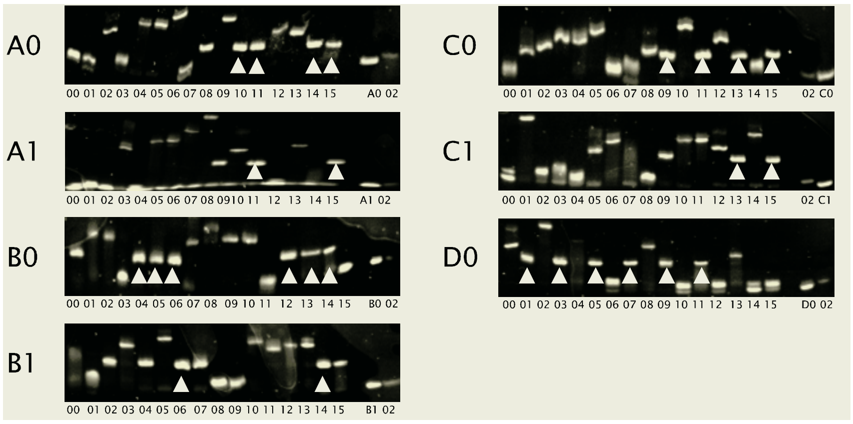

A0
00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 A0 02
A1
00 01 02 03 04 05 06 07 08 0910 11 12 13 14 15 A1 02
B0
00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 B0 02
B1
00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 B1 02
C0
00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 02 C0
C1
00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 02 C1
D0
00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 D0 02

Fig. 2. Heteroduplex migration assay for all blocker/library combinations. Each gel contains the complete library (00-15) of oligonucleotides hybridized to the indicated blocker. The rightmost two lanes were loaded with unhybridized blocker and library 02. Arrows indicate apparent homoduplexes, which can be identified by coinciding fluorescence in both detection channels. The grayscale images in this figure only show total fluorescence

Fig. 2 shows the gel images for all combinations of blockers with library molecules. These images are grayscale superpositions of the 530 BP (showing the blocker fluorescein label) and 610 LP (library Cy5) channels. In RGB stack images, duplexes appear as yellow bands, since they fluoresce in both channels at the same location. Red (Cy5) and green (fluorescein) bands indicate non-hybridizing oligonucleotides. Arrows indicate apparent homoduplexes that can be identified from colour images.

Every blocker should only be able to form a perfect duplex with one of the library oligonucleotides, but Fig. 2 shows up to 6 apparent homoduplexes per blocker. No improvement was found using MD gel matrix or longer gels (not shown). Nonetheless, some solutions to the satisfiability problem can be identified from Fig. 2. Library oligos 00, 02 and 08 (*abcd* = {0000}, {0010} and {1000}, respectively) do not behave as a homoduplex in any combination.

3.2 Mismatch Endonucleases

Fig. 3 shows the effects of T7EI and CEL I on homoduplex and heteroduplex DNA. Both were incubated for a range of times. In our hands, the T7EI enzyme did not have any discernable effect on any DNA sample (here, 0.2 units were used per reaction; 1 unit per reaction gave identical results), and was therefore not considered for further experiments.

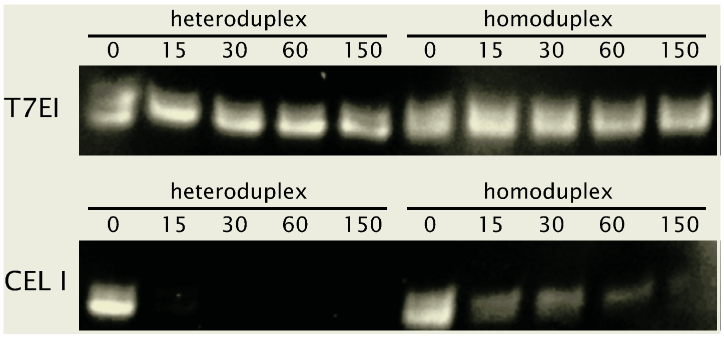


Fig. 3. T7EI & CEL I time series. 3 pmol samples of heteroduplex (C0 + 11) or homoduplex DNA (C0 + 13) were subjected to both endonucleases for up to 150 minutes. Samples were analyzed on a 12% denaturing gel. Images show total (combined 530 BP and 610 LP) fluorescence

CEL I, however, has a clear effect on all samples. The enzyme completely degrades the mismatched DNA within 30 minutes. Perfect dsDNA, although also subject to degradation, is still detectable after 150 minutes of reaction. ssDNA is quickly degraded completely (not shown). To test whether CEL I would successfully pick blocking from non-blocking combinations, all library molecules were incubated with blockers and enzyme (Fig. 4). From these results, satisfying assignments could be identified (summarized in table 2).

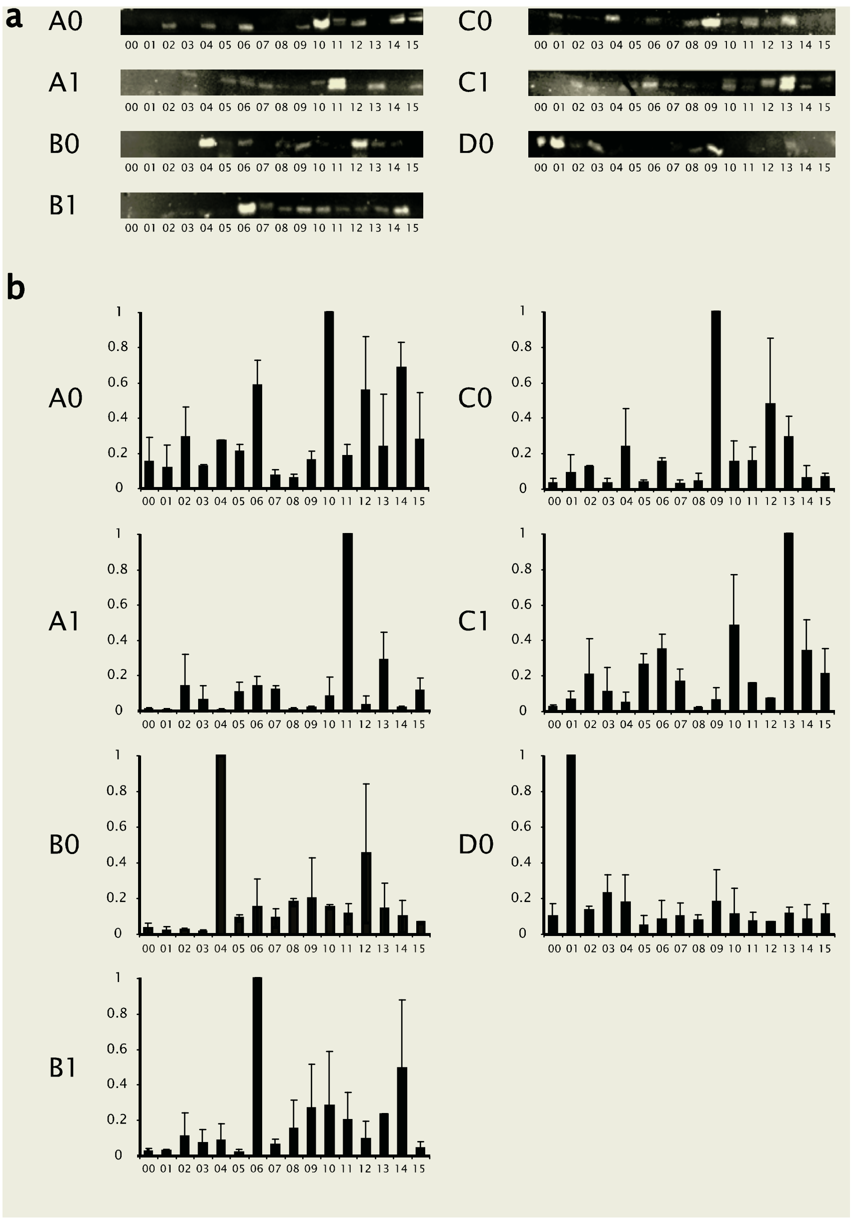


Fig. 4. a Effects of CEL I on all blocker/library combinations. Shown are denaturing gels of complete sets of library oligonucleotides and blockers, incubated with CEL I. **b** Quantified fluorescence from the gels. Fluorescence signals from the 530 BP channel are given relative to the signal of the most intense band on each gel (set to 1). The y-axis shows relative fluorescence, the x-axis the library molecules. Values are averages of two independent experiments. Error bars give standard deviations

Table 2. Apparent solutions to F (eq. 1)

Library molecule	*abcd*	falsified by duplex migration	falsified by CEL I
00	0000		
01	0001	D0	D0
02	0010		
03	0011	D0	
04	0100	B0	B0
05	0101	B0, D0	
06	0110	B0, B1	B1
07	0111	D0	
08	1000		
09	1001	C0, D0	C0
10	1010	A0	A0
11	1011	A0, A1, C0, D0	A1
12	1100	B0	
13	1101	B0, C0, C1	C1
14	1110	A0, B0, B1	
15	1111	A0, A1, C0, C1	

4 Discussion

Differential duplex migration did not provide a suitable test system to distinguish every satisfying solution from non-satisfying ones. There is no general theory describing the effect of anomalous DNA conformations on migration rate, and it was already known that not all mismatches can be detected this way [16,17]. A possible explanation for the ambiguous results reported here is the length of the DNA molecules: heteroduplex migration is generally recommended for DNA 100 – 500 bp in length. Such lengths also accentuate the effect of a single mismatch, which produces a bend in the helix. In addition, the nature of the mismatches studied here may have contributed. A single variable difference between blocker and library is represented by 4 non-matching basepairs at a molecular level. This will probably form a bubble-type mismatch, which may not always be subject to higher gel resistances.

We believe that with careful optimization of the encoding, the use of longer molecules (perhaps in combination with scaling to larger problem instances) and more sophisticated analytical techniques (e.g. capillary electrophoresis), the method holds considerable promise. In particular, duplex migration might be employed as a phenotype for the implementation of evolutionary algorithms in DNA [18].

The CEL I assay gave more consistent results. However, the results are difficult to interpret from visual inspection of single gels, because CEL I also degrades perfect duplexes. This breakdown of homoduplex DNA may be due to equilibrium fraying of the molecules, continuously giving the enzyme a toehold on the duplex. Therefore, longer molecules may also be an option for this method.

Using the blocking algorithm and encoding as reported here, the mismatch endonuclease assay is only useful as an analytical method. Because satisfying library molecules are fragmented, multiple rounds of selection (as in evolutionary algorithms) cannot be easily implemented. However, several other proteins that bind mismatches (such as MutS [19]) do not destroy the DNA molecule. In future experiments, such proteins may be used in a gel-shift assay [20]. Besides the enzymatic method tested here, chemical cleavage of mismatches [21] could be considered. In conclusion, we believe that the use of mutation detection techniques is an interesting option for DNA based computing.

Acknowledgments

We thank Dr Edwin Cuppen for the kind gift of purified CEL I enzyme. This work was supported by the Netherlands Organization for Scientific Research (NWO).

References

1. Faulhammer, D., Cukras, A. R., Lipton, R. J., Landweber, L. F., Molecular computation: RNA solutions to chess problems. Proc. Natl. Acad. Sci. USA **97** (2000) 1385-1389
2. Liu, Q. H., Wang, L. M., Frutos, A. G., Condon, A. E., Corn, R. M., Smith, L. M., DNA computing on surfaces. Nature **403** (2000) 175-179
3. Braich, R. S., Chelyapov, N., Johnson, C., Rothemund, P. W. K., Adleman, L., Solution of a 20-Variable 3-SAT problem on a DNA computer. Science **296** (2002) 499-502
4. Rozenberg, G., Spaink, H., DNA Computing by blocking. Theoretical Computer Science **292** (2003) 653-665
5. Schmidt, K. A., Henkel, C. V., Rozenberg, G., Spaink, H. P., DNA computing using single molecule hybridization detection. Nucl. Acids Res., in press
6. Taylor, G. R., Enzymatic and chemical cleavage methods. Electrophoresis **20** (1999) 1125-1130
7. Kristensen, V. N., Kelefiotis, D., Kristensen, T., Borresen-Dale, A. L., High-throughput methods for detection of genetic variation. Biotechniques **30** (2001) 318-332
8. Nataraj, A. J., Olivos-Glander, I., Kusukawa, N., Highsmith, W. E., Single-strand conformation polymorphism and heteroduplex analysis for gel-based mutation detection. Electrophoresis **20** (1999) 1177-1185
9. Mashal, R. D., Koontz, J., Sklar, J., Detection of mutations by cleavage of DNA heteroduplexes with bacteriophage resolvases. Nat. Genet. **9** (1995) 177-183
10. Oleykowski, C. A., Bronson Mullins, C. R., Godwin, A. K., Yeung, A. T., Mutation detection using a novel plant endonuclease. Nucl. Acids. Res. **26** (1998) 4597-4602
11. Yang, B., Wen, X., Kodali, N. S., Oleykowski, C. A., Miller, C. G., Kulinski, J., Besack, D., Yeung, J. A., Kowalski, D., Yeung, A. T., Purification, cloning, and characterization of the CEL I nuclease. Biochemistry **39** (2000) 3533-3541
12. Loakes, D., The applications of universal DNA base analogues. Nucl. Acids Res. **29** (2001) 2437-2447
13. SantaLucia, J., A unified view of polymer, dumbbell, and oligonucleotide DNA nearest-neighbor thermodynamics. Proc. Natl. Acad. Sci. USA **95** (1998) 1460-1465
14. Peyret, N., Seneviratne, P. A., Allawi, H. T., SantaLucia, J., Nearest-neighbor thermodynamics and NMR of DNA sequences with internal A - A, C - C, G - G, and T - T mismatches. Biochemistry **38** (1999) 3468-3477

15. Ganguly, A., Rock, M. J., Prockop, D. J., Conformation-sensitive gel electrophoresis for rapid detection of single- base differences in double-stranded PCR products and DNA fragments: evidence for solvent-induced bends in DNA heteroduplexes. Proc. Natl. Acad. Sci. USA **90** (1993) 10325-10329
16. Highsmith, W. E., Jin, Q., Nataraj, A. J., O'Connor, J. M., Burland, V. D., Baubonis, W. R., Curtis, F. P., Kusukawa, N., Garner, M. M., Use of a DNA toolbox for the characterization of mutation scanning methods. I: construction of the toolbox and evaluation of heteroduplex analysis. Electrophoresis **20** (1999) 1186-1194
17. Upchurch, D. A., Shankarappa, R., Mullins, J. I., Position and degree of mismatches and the mobility of DNA heteroduplexes. Nucl. Acids Res. **28** (2000) e69
18. Wood, D., Chen, J., Antipov, E., Lemieux, B., Cedeno, W., A DNA implementation of the Max 1s problem. In: Banzhaf, W., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., Smith, R. E. (eds.), Proceeding of the genetic and evolutionary computation conference 1999. Morgan Kaufman, San Francisco (1999) 1835-1841
19. Brown, J., Brown, T., Fox, K. R., Affinity of mismatch-binding protein MutS for heteroduplexes containing different mismatches. Biochem. J. **354** (2001) 627-633
20. Goode, E., Wood, D. H., Chen, J., DNA Implementation of a Royal Road Fitness Evaluation. In: Condon, A. , Rozenberg, G. (eds.), DNA computing, proceedings 6th international meeting on DNA based computers. Springer-Verlag, Berlin Heidelberg (2001) 247-262
21. Bui, C. T., Rees, K., Lambrinakos, A., Bedir, A., Cotton, R. G. H., Site-selective reactions of imperfectly matched DNA with small chemical molecules: applications in mutation detection. Bioorg. Chem. **30** (2002) 216-232

Bond-Free Languages: Formalizations, Maximality and Construction Methods

Lila Kari[1], Stavros Konstantinidis[2,⋆], and Petr Sosík[1,3]

[1] Department of Computer Science, The University of Western Ontario, London, ON, Canada, N6A 5B7
{lila, sosik}@csd.uwo.ca

[2] Dept. of Mathematics and Computing Science, Saint Mary's University, Halifax, Nova Scotia, B3H 3C3 Canada
s.konstantinidis@stmarys.ca

[3] Institute of Computer Science, Silesian University, Opava, Czech Republic

Abstract. The problem of negative design of DNA languages is addressed, that is, properties and construction methods of large sets of words that prevent undesired bonds when used in DNA computations. We recall a few existing formalizations of the problem and then define the property of sim-bond-freedom, where sim is a similarity relation between words. We show that this property is decidable for context-free languages and polynomial-time decidable for regular languages. The maximality of this property also turns out to be decidable for regular languages and polynomial-time decidable for an important case of the Hamming similarity. Then we consider various construction methods for Hamming bond-free languages, including the recently introduced method of templates, and obtain a complete structural characterization of all maximal Hamming bond-free languages. This result is applicable to the θ-k-code property introduced by Jonoska and Mahalingam.

1 Introduction

Most of the operations involved in DNA computations rely on the capability of controlling the *bonds* that can be formed between *(single-stranded) DNA molecules*. Such bonds are created due to the well-known Watson-Crick *complementarity* property of the four nucleotides A, C, G, T, which are the building blocks of DNA molecules. This property is important in conjunction with the fact that every molecule has a certain *orientation*, which is denoted by placing the symbols '$5'-$' and '$-3'$' at the two ends of the sequence of nucleotides comprising the molecule. For example, the molecules $5'-ACCGT-3'$ and $3'-ACCGT-5'$ are different – they have different chemical properties. In practice, the collection of DNA molecules exists as a 'soup' inside a test *tube*. Under favorable physical

⋆ Corresponding author.

tube conditions, if a molecule of the form $5' - X_1X_2 \cdots X_k - 3'$, where each X_i is a nucleotide, encounters the molecule $5' - Y_k \cdots Y_2Y_1 - 3'$ in which each nucleotide Y_i is the complement of X_i, then the pairs (X_i, Y_i) will form k chemical bonds and a double-stranded structure will be created – see Figure 1(a).

```
5'- A G T T C C -3'    5'- v A G T T C C w -3'    5'- v A G T T C C w
    | | | | | |              | | | | | |                  | | | | | |  x
3'- T C A A G G -5'    3'- z T C A A G G y -5'    3'- z T C A A G G y
       (a)                      (b)                        (c)
```

Fig. 1. Vertical bars represent bonds between complementary nucleotides. In (b), the complementary parts $5' - AGTTCC - 3'$ and $5' - GGAACT - 3'$ of the DNA molecules $5' - vAGTTCCw - 3'$ and $5' - yGGAACTz - 3'$ bind together. In (c), the molecule $5' - vAGTTCCwxyGGAACTz - 3'$ is twisted at x and its complementary parts bind together

It is important to note that bonds can be formed even between complementary *parts* of two molecules, provided that these parts are *sufficiently long* – see Figure 1(b). Moreover, a molecule containing two complementary parts can bind to itself, or to a copy of itself – see Figure 1(c).

The bonds shown in Figure 1 are formed between parts that are *perfect* complements of each other. In practice, however, it is possible that two parts of molecules will bind together even if some of their corresponding nucleotides are not complementary to each other – see Figure 2.

```
5'- v A G A T T C C G T G w -3'       5'- v A G A T T C C G T G w
      | | |   | | |   | |                   | | |   | | |   | |  x
3'- z T C T C A G G A A C y -5'       3'- z T C T C A G G A A C y
             (a)                                   (b)
```

Fig. 2. In (a), parts of two DNA molecules bind together although these parts are not perfect complements of each other. In (b), the same parts appear in one molecule

1.1 The Problem of Undesirable Bonds

The success of a DNA operation relies on the assumption that no accidental bonds can be formed between molecules in the tube before the operation is initiated, or even during the operation. With this motivation, one of the foremost problems in DNA computing today is the following.

Problem 1. Define a large, potential collection of DNA molecules such that there can be no (sufficiently long and possibly imperfect) complementary parts in any two molecules, and no (sufficiently long and possibly imperfect) complementary parts in any one molecule.

In many cases in the literature, this problem is addressed in conjunction with the *uniqueness* problem, which involves designing molecules whose parts are different between each other. The motivation here is that, usually, a DNA operation is intended only for molecules containing a specific pattern (or specific patterns) of nucleotides. In this paper, however, we focus on Problem 1.

1.2 Notation for Molecules and Bonds

We proceed now with establishing the notation that would allow us to describe formalizations of Problem 1. Specifically, we define the terms *word, subword, language, involution, and codeword.*

A given alphabet can be used to form sequences of symbols that are called *words.* For example, 01001 is a word over the alphabet $\{0, 1\}$. The *length* of a word w is denoted by $|w|$. For example, $|01001| = 5$. The prime example of an alphabet will be the DNA alphabet $\{A, C, G, T\}$. In this case, we agree that the left end of a *DNA word* represents the $5'$−end of the corresponding DNA molecule. For example, the word $CCATGT$ represents the molecule $5' - CCATGT - 3'$. If a word w can be written in the form xyz – this is the catenation of some words x, y and z – then we say that y is a *subword* of w. A *language* is any set of words. We shall use the expression 'x is a subword of a language' as a shorthand for x is a subword of some word in the language. One use of a language L is to represent all the possible distinct copies of DNA molecules that might appear in a tube. In this case, we refer to L as a *tube language* and we assume that every word in L is of length at least k, for some *parameter* k. This parameter represents the smallest length of two molecule parts for which it is possible to form a stable bond.

To represent the complementarity of nucleotides we use the concept of antimorphic involution introduced in [13]. In general an *involution* of an alphabet Σ is a function $\theta : \Sigma \to \Sigma$ such that $\theta(\theta(a)) = a$, for all symbols a in Σ. The involution is called *antimorphic* if we extend it to words such that $\theta(a_1 \cdots a_n) = \theta(a_n) \cdots \theta(a_1)$, where each a_i is a symbol in Σ. The prime example of an antimorphic involution will be the *DNA involution* τ such that

$$\tau(A) = T,\ \tau(T) = A,\ \tau(C) = G,\ \tau(G) = C.$$

For example, $\tau(ACCGTT) = AACGGT$. In general, for two DNA words x and y of length k, *the identity* $\tau(x) = y$ *represents the fact that the molecules (or parts of molecules)* $5' - x - 3'$ *and* $5' - y - 3'$ *could bind to each other.* According to the requirement in Problem 1, if $k = 6$, the words $ACCGTT$ and $AACGGT$ should not be subwords of the tube language L.

In the literature on DNA encodings, the tube language L is usually equal to, or a subset of, K^+, where K is a finite language whose elements are called *codewords.* The language K^+ consists of all words that are obtained by concatenating one or more codewords from K. For a nonnegative integer n, the notation K^n is used for the set of all words that are obtained by concatenating any n codewords from K. In general, K might contain codewords of different lengths.

In many cases, however, the set K consists of words of a certain fixed length l. In this case, we shall refer to K as a *code of length l*.

1.3 Formalizations of the Problem of Undesirable Bonds

With the preceding terminology in mind, Problem 1 is called the negative word design problem in [18]. Now we recall a few existing formalizations of Problem 1 and we propose a new one, which appears to be closer to the intuition behind the problem. It should be noted, however, that all formalizations are inter-related in some interesting ways.

One of the most recent attempts to address Problem 1 appears in [10]. In that paper, the authors require that a tube language L must satisfy the following property.

P1$[k]$**:** If x and y are any subwords of L of length k then $x \neq \tau(y)$.

A language satisfying this property is called a τ-k-code in [10]. An advantage of this formalization is that the property is defined *independently* of the structure of L. This property is also considered implicitly in [3] and [6]. In particular, reference [3] considers tube languages of the form $(sZ)^+$ satisfying **P1**$[k]$, where s is a fixed word of length k and Z is a code of length k – the notation sZ represents the set of all words sz such that z is in Z.

In [9], the authors introduce the concept of a *strictly τ-free code K*, which is a generalization of the notion of comma-free code [12], and show that the language K^+ must be strictly τ-free as well. Here we shall assume that K is of fixed length k. In this formalization the tube language L is equal to K^+. Using the tools of [9], it can be shown that L is a strictly τ-free language *iff* (if and only if) L satisfies the following property

P2$[k]$**:** If x is a subword of L of length k and v is a codeword in K then $x \neq \tau(v)$.

We note that similar properties are considered also in [15] and [16].

As noted earlier, parts of DNA molecules can bind to each other even if they are not perfect complements of each other. Hence, although sufficient, the condition $\tau(x) = y$ might not be necessary for the DNA words x and y to stick together. The common approach to deal with this is to modify the above condition by using the Hamming distance function $H(\cdot, \cdot)$. More specifically, for two words x and y of length k, *the relation $H(x, \tau(y)) \leq d$ represents the fact that the molecules (or parts of molecules) $5' - x - 3'$ and $5' - y - 3'$ could bind to each other.* Here, d is a nonnegative integer less than k.

In [5] and [21], the authors consider codes K of length k satisfying the following property

P3$[d, k]$**:** If u and v are any codewords in K then $H(u, \tau(v)) > d$.

In fact the above property is studied in conjunction with the uniqueness property $H(K) > d$.

Reference [2] considers a measure between two words, which is applied to codes of length k whose words can be concatenated in arbitrary ways. Thus, the tube language here is $L = K^+$. The code K satisfies certain uniqueness conditions as well as conditions related to Problem 1. In particular, the tube language $L = K^+$ satisfies the following property.

P4$[d, k]$**:** If x is a subword of L of length k and v is a codeword in K then $H(x, \tau(v)) > d$.

We note that also reference [19] considers this property for tube languages of the form $K_1K_2 \cdots K_m$, where each K_i is a certain code of length k.

With '$H(x, \tau(y)) \leq d$' as the criterion for x and y to bind together, it appears that **P4**$[d, k]$ is the strictest property in the literature for addressing Problem 1. This property, however, is not sufficient in general for avoiding undesirable bonds in the tube. To see this, consider the case where

$$d = 1, \quad k = 5, \quad K = \{ACGAT, CCGAA\}.$$

One can verify that the language K^+ satisfies **P4**$[d, k]$ and that the DNA words $ACGATACGATCCGAA$ and $ACGATCCGAACCGAA$ are in K^+ and contain the subwords $GATCC$ and $CGATC$ such that

$$H(GATCC, \tau(CGATC)) \leq 1.$$

Motivated by the above observation, we introduce the following property of a tube language L.

P5$[d, k]$**:** If x and y are any subwords of L of length k then $H(x, \tau(y)) > d$.

Note that, as in the case of **P1**$[k]$, the new property is defined independently of the structure of L. Any tube language satisfying this property will be called a $(\tau, H_{d,k})$*-bond-free language*.

We list now a few interesting connections among the properties **P1**–**P5**. We note that the condition $x \neq \tau(y)$ is equivalent to $H(x, \tau(y)) > 0$.

P3 and P5: In this paper we introduce the *subword closure operation* $\otimes$ and we show that if a code K satisfies **P3**$[d, k]$ then the language $K^{\otimes}$ satisfies **P5**$[d, k]$.

P4 and P2: It is evident that any language K^+ satisfying **P4**$[d, k]$ also satisfies **P2**$[k]$. Moreover, **P4**$[0, k]$ is identical to **P2**$[k]$. We can show that, for every code Q of length q, if the language Q^+ satisfies **P2**$[q]$ then the language $(Q^{d+1})^+$ satisfies **P4**$[d, q(d + 1)]$, for any $d > 0$.

P4 and P5: It is evident that any language K^+ satisfying **P5**$[d, k]$ also satisfies **P4**$[d, k]$. Moreover, it can be shown that if K^+ satisfies **P4**$[d, k]$ then $(K^2)^+$ satisfies **P5**$[d, k]$.

P5 and P1: Obviously, any language satisfying **P5**$[d, k]$ also satisfies **P1**$[k]$. Moreover, the property **P1**$[k]$ coincides with **P5**$[0, k]$. It can be shown that every language satisfying **P1**$[q]$, for some positive integer q, also satisfies **P5**$[d, q(d+1)]$ for every $d > 0$, and conversely, if the language is of the form K^+ and satisfies **P5**$[d, k]$ then it satisfies **P1**$[k - d]$ as well.

Important Note. Proofs of the results obtained in this paper can be found in the full version [17].

1.4 A More General Formalization: (θ, sim)-Bond-Freeness

The choice of the Hamming distance in the condition '$H(x, \tau(y))$' for *similarity* between words is a very natural one and has attracted a lot of interest in the literature. One might argue, however, that parts of two DNA molecules could form a stable bond even if they have different lengths – hybridizations of this type are addressed in [1]. Based on this observation, the condition for two subwords x and y to bind together should be

$$|x|, |y| \geq k \quad \text{and} \quad Lev(x, \tau(y)) \leq d.$$

The symbol $|u|$ denotes the length of the word u and $Lev(u, v)$ is the *Levenshtein distance* between the words u and v – this is the smallest number of substitutions, insertions and deletions of symbols required to transform u to v. With this formulation, the condition for similarity based on the Hamming distance can be rephrased as follows

$$|x|, |y| \geq k \quad \text{and} \quad H(x, \tau(y)) \leq d,$$

where we assume that $H(u, v) = \infty$ if the words u and v have different lengths. In general, for any similarity relation $\mathrm{sim}(\cdot, \cdot)$ between words and for every involution θ, we define the following property of a language L.

$\mathbf{P}[\theta, \mathrm{sim}]$: If x and y are any nonempty subwords of L then $\mathrm{sim}(x, \theta(y))$ is false.

Any language satisfying $\mathbf{P}[\theta, \mathrm{sim}]$ is called a (θ, sim)-*bond-free language.*

The precise definition of a similarity relation is given in Section 2. It can be shown that the relations '$|u|, |v| \geq k$ and $H(u, v) \leq d$' and '$|u|, |v| \geq k$ and $Lev(u, v) \leq d$' are indeed similarity relations. *For these relations we shall use the notation $H_{d,k}$ and $Lev_{d,k}$, respectively.*

2 Decidability Questions About (θ, sim)-Bond-Freedom

In this section, we use the general tools about language operations and trajectories obtained in [16] and we show that one can decide in quadratic time whether a given regular language is (θ, sim)-bond-free. Moreover, we show that this problem is decidable even when the given language is context-free. Then, we use also the general tools about maximal solutions of language inequations developed in [14] and [16] to establish the decidability of whether a given regular language is maximal with respect to the (θ, sim)-bond-free property. The acronyms *DFA* and *NFA* stand for deterministic and nondeterministic, respectively, finite automaton. A relation between words (binary relation) is *rational* if it is realized by a finite-state transducer.

Definition 1. *A binary relation* sim *is called a* similarity relation with parameters (t, l), *where t and l are nonnegative integers, if the following conditions are satisfied. (i) If* $\mathrm{sim}(u, v)$ *is true then* $\mathrm{abs}(|u| - |v|) \leq t$. *(ii) If* $\mathrm{sim}(u, v)$ *is true and $|u|, |v| > l$ then there are proper subwords x and y of u and v, respectively, such that* $\mathrm{sim}(x, y)$ *is true.*

We can interpret the above conditions as follows: (i) the lengths of two similar words cannot be too different and (ii) if two words are similar and long enough, then they contain two similar proper subwords. *In the rest of the section we shall assume that* sim *is a fixed, but arbitrary, similarity relation with parameters* (t, l). It is evident that the relation $H_{d,k}$ defined in Subsection 1.4 is an example of a similarity relation with parameters $(0, k)$. It can be shown that also $Lev_{d,k}$ is a similarity relation, with parameters $(d, d + k)$ – see [17].

Theorem 1. *Assume that* sim *is a rational relation. The following problem is decidable in quadratic time.* ***Input:*** *NFA A.* ***Output:*** *Y/N, depending on whether $L(A)$ is a (θ, sim)-bond-free language.*

We note that for most of the DNA language properties considered in [9, 15, 16] the above problem is undecidable for context-free languages. As the $(\theta$, sim$)$-bond-free property seems to be rather general, it might be surprising that the same problem is decidable.

Theorem 2. *If the similarity relation* sim *is computable, then it is decidable whether a given context-free language is* $(\theta$, sim$)$*-bond-free.*

Corollary 1. *Let d and k be nonnegative integers with $k \geq 1$. It is decidable whether a given context-free language is $(\theta, H_{d,k})$-bond-free (or $(\theta, Lev_{d,k})$-bond-free).*

Theorem 3. *Assume that the similarity relation* sim *is rational. Then the following problem is decidable.* ***Input:*** *NFAs A and B such that $L(A)$ is a (θ, sim)-bond-free subset of $L(B)$.* ***Output:*** *Y/N, depending on whether $L(A)$ is a maximal (θ, sim)-bond-free subset of $L(B)$.*

3 Decidability of Maximality in the Hamming Case

In the literature on DNA encodings, and in coding theory in general, the set of words that are involved in the application of interest are usually formed by concatenating shorter words of a certain fixed length. Following this practice, we consider languages that are subsets of $(\Sigma^k)^+$, for some positive integer k. We call such languages *k-block languages*. Naturally, any regular k-block language can be represented by a special type of lazy DFA, which we call *k-block DFA*. This is a deterministic finite automaton such that, for every production $pu \to q$, the word u is of length k.

The decision method for maximality of the previous section is not of polynomial time. In this section, however, we are able to show a polynomial time algorithm for testing whether a given regular langauge is $(\theta, H_{d,k})$-bond-free, for $d = 0$ or $d = 1$. We remind the reader that, in the case of $d = 0$, the property coincides with $\mathbf{P1}[k]$ – see Subsection 1.3. Next we illustrate the concept of maximality with an example. *The notation* $\mathrm{Sub}_k(L)$ *represents the set of all subwords of length k of the language L.*

Example 3.1. Consider the code $K_1 = \{AA, AC, CA, CC\}$ over the DNA alphabet and the 2-block language K_1^+. Let $S_1 = \mathrm{Sub}_2(K_1^+)$. Then, S_1 is equal to K_1 and $S_1 \cap \tau(S_1)$ is empty. Hence, the language K_1^+ is a $(\tau, H_{0,2})$-bond-free subset of $(\Sigma^2)^+$. Moreover, there is no word v in $\Sigma^2 - K_1$ such that the language $(K_1 \cup \{v\})^+$ is $H_{0,2}$-bond-free. For example, if $v = AG$ then GC would be a subword of length 2 of $(K_1 \cup \{v\})^+$ such that $GC = \tau(GC)$. On the other hand, it is possible to add AG as a subword with the constraint that AG cannot be followed by CA or CC. In fact we can add also GA as a subword, provided that GA cannot be preceded by AC, CC, or AG. More specifically, consider the language L_2 accepted by the 2-block DFA $(\Sigma, \{1,2,3,4\}, 1, \{2,3,4\}, P_2)$, where $2, 3, 4$ are the final states and the set of productions P_2 is equal to

$$\{1u \to 2,\ 1v \to 3,\ 1(AG) \to 4 \mid u = AA, CA, GA \text{ and } v = AC, CC\} \cup$$
$$\{2u \to 2,\ 2v \to 3,\ 2(AG) \to 4 \mid u = AA, CA, GA \text{ and } v = AC, CC\} \cup$$
$$\{3u \to 2,\ 3v \to 3,\ 3(AG) \to 4 \mid u = AA, CA \text{ and } v = AC, CC\} \cup$$
$$\{4(AG) \to 4,\ 4(AC) \to 3,\ 4(AA) \to 2\}.$$

The language L_2 is a proper superset of K_1^+ and is a $(\tau, H_{0,2})$-bond-free subset of $(\Sigma^2)^+$. In fact it can be shown that L_2 is maximal with this property [17].

Theorem 4. *Let d be a fixed value in $\{0,1\}$. The following problem is computable in polynomial time.* ***Input:*** *k-block DFA A such that $L(A)$ is a $(\theta, H_{d,k})$-bond-free subset of $(\Sigma^k)^+$.* ***Output:*** *Y/N, depending on whether $L(A)$ is maximal with that property. Moreover, if $L(A)$ is not maximal, output a minimal-length word $w \in (\Sigma^k)^+ - L(A)$ such that $L(A) \cup \{w\}$ is a $(\theta, H_{d,k})$-bond-free subset of $(\Sigma^k)^+$.*

4 Construction Methods for the Hamming Case

In this section we describe methods for constructing $(\tau, H_{d,k})$-bond-free languages. We focus on languages that are subsets of $(\Sigma^k)^+$ or $\Sigma^k\Sigma^*$. *We assume throughout that k and d are integers, with $k \geq 1$ and $0 \leq d < k$, and τ is the DNA involution.* Moreover, we introduce the subword closure operation, $\otimes$, which plays an important role in the sequel.

Let d be a nonnegative integer and let S and S_1 be languages containing only words of the same length k, for some positive integer k. The *Hamming ball* $H_d(S)$ of S is the set $\{v \mid H(v,z) \leq d$, for some $z \in S\}$. Note that $H_d(S) = S$ when $d = 0$. The *subword closure* $S^\otimes$ of S is the set $\{w \in \Sigma^* \mid |w| \geq k,\ \mathrm{Sub}_k(w) \subseteq S\}$. We note that $S_1 \subseteq S$ iff $S_1{}^\otimes \subseteq S^\otimes$. This implies that, if $S_1 \neq S$ then $S_1{}^\otimes \neq S^\otimes$. Moreover we note that, given S, one can construct in linear time a DFA accepting $S^\otimes$ [17].

4.1 Direct Methods

Here we consider analytical methods that do not rely on previously constructed languages. The first method is based on the concept of a template and the operation '$\cdot$': $0 \cdot 0 = C$, $0 \cdot 1 = G$, $1 \cdot 0 = T$, $1 \cdot 1 = A$ [2]. This operation is extended to binary words of the same length in a natural manner. A *k-template* is any binary word of length k. If x is a k-template and E is subset of $\{0,1\}^k$ then $x \cdot E = \{x \cdot v \mid v \in E\}$. The construction method of [2] involves choosing a k-template x and a code E such that $x \cdot E$ satisfies a desired property. In our case, we are interested in k-templates x such that

$$H(x_2 x_1, (x_4 x_3)^R) > d, \tag{1}$$

for all prefixes x_1 and x_3 of x and suffixes x_2 and x_4 of x.

Theorem 5. *Let x be a k-template satisfying (1). Then the language $(x \cdot \{0,1\}^k)^+$ is $(\tau, H_{d,k})$-bond-free.*

Observe that the cardinality of the code $x \cdot \{0,1\}^k$ is 2^k. The advantage of the method of templates is that properties of the template x, which is a simple object, are passed gracefully to the code $x \cdot E$, where E is any subset of $\{0,1\}^k$. We note that many of the templates listed in [2] satisfy (1).

We introduce now another direct construction method. The bond-free language is again of the form K^+, where K is a code of fixed-length. Moreover there is a set I of positions in which the codeword symbols are always in $\{A, C\}$. The method is described more formally in the next theorem. The notation $k \,\%\, 2$ stands for the remainder of the integer division $k/2$, and $v[i]$ stands for the symbol of the word v at position i.

Theorem 6. *Let I be a nonempty subset of $\{1, \ldots, k\}$ of cardinality $\lfloor k/2 \rfloor + 1 + \lfloor (d + k \,\%\, 2)/2 \rfloor$. Then the language K^+ is $(\tau, H_{d,k})$-bond-free, where*

$$K = \{v \in \Sigma^k \mid \text{ if } i \in I \text{ then } v[i] \in \{A, C\}\}.$$

Let l be the quantity $\lfloor k/2 \rfloor + 1 + \lfloor (d + k \,\%\, 2)/2 \rfloor$ that appears in the above theorem. The size of the code K is $2^l 4^{k-l}$. On the other hand the method of k-templates produces codes K of size 2^k. Obviously, $2^l 4^{k-l} \geq 2^k$. Moreover, one can verify that $k = l$ iff d is in $\{k-2, k-3\}$. An advantage of the method of Theorem 6 is that we can construct $(\tau, H_{d,k})$-bond-free languages with a large ratio d/k. Another advantage of some codes K defined in the previous theorem is that one can encode and decode information in linear time [17].

4.2 Methods Based on the Catenation Closure

The main idea here is that the catenation closure of Q^{d+1}, that is the language $(Q^{d+1})^+$, is $(\tau, H_{d,k})$-bond-free if Q is of length q with the property that Q^+ is $(\tau, H_{0,q})$-bond-free. The correctness of the method is based on the following theorem.

Theorem 7. *Let j and q be positive integers and let L be a subset of $\Sigma^{jq}\Sigma^*$. If L is $(\tau, H_{t,q})$-bond-free, for some integer $t \geq 0$, then it is also $(\tau, H_{d,k})$-bond-free, where $d = j(t+1) - 1$ and $k = jq$.*

Observe that for $t = 0$, the above theorem says that nearly every language that is $(\tau, H_{0,q})$-bond-free is *inherently* $(\tau, H_{d,k})$-bond-free for any $d > 0$ and any $k \geq q(d+1)$. This is a connection between the properties **P1** and **P5** considered in Section 1.

With the notation of the above theorem, let Q be a code of length q such that the language Q^+ is $(\tau, H_{t,q})$-bond-free. Let $K = Q^j$ and let $k = jq$. The code Q could be defined by some direct method, or by brute force for small values of q and t. In either case, the language K^+ is $(\tau, H_{d,k})$-bond-free.

In the case of $t = 0$, we have that $j = d + 1$ and the cardinality of the code K is $|Q|^{d+1}$, which can be larger than the cardinality of the codes defined in Theorem 6 with the same parameters. For example, if $Q = \{A, C\}^2 \Sigma \cup \{A, C\}\{G, T\}\{A, C\}$, then the code $K = Q^{d+1}$ consists of 24^{d+1} codewords. On the other hand, if the code K is defined using Theorem 6 for $k = 3(d + 1)$ then the cardinality of K is equal to 16^{d+1}.

The following observation can be viewed as a converse type of Theorem 7.

Theorem 8. *Let K be any set of words such that the language K^+ is $(\tau, H_{d,k})$-bond-free, for some integers $d \geq 0$ and $k \geq 1$. Then the language K^+ is also $(\tau, H_{0,k-d})$-bond-free.*

4.3 All Maximal (Hamming) Bond-Free Languages

With the results of Section 3 in mind [17], we understand that the languages of the form K^+ obtained by the preceding methods are not necessarily maximal. In what follows we discuss methods of obtaining new bond-free languages, possibly maximal, from old ones using the subword closure operation $\otimes$. We also need the following, slightly restricted, version of the subword closure of S, where S is any code of fixed length k, $S^{\oplus} \triangleq S^{\otimes} \cap (\Sigma^k)^+$. We call $S^{\oplus}$ the *block closure* of S.

Theorem 9. *Let S be a set of words of fixed length k. Then each of the languages $S^{\otimes}$ and $S^{\oplus}$ is $(\tau, H_{d,k})$-bond-free iff*

$$\tau(S) \cap H_d(S) = \emptyset. \tag{2}$$

Using the above observation we can extend $(\tau, H_{d,k})$-bond-free languages of the form K^+, such as those constructed earlier, as follows – we assume the words of K are of fixed length k. Let $S = \mathrm{Sub}_k(K^2) = \mathrm{Sub}_k(K^+)$. Then S satisfies (2) and, therefore, the language $S^{\otimes}$ is a $(\tau, H_{d,k})$-bond-free language that includes K^+. Next consider any code K of length k satisfying property **P3**$[d, k]$ – recall from Section 1 that such codes have been studied in [5] and [21]. Using again the above theorem it follows that $K^{\otimes}$ is a $(\tau, H_{d,k})$-bond-free language.

The question that arises now is when the bond-free languages of Theorem 9 are maximal. The following result addresses this question. In fact we show a complete structural characterization of all maximal $(\tau, H_{d,k})$-bond-free subsets of $(\Sigma^k)^+$ and $\Sigma^k \Sigma^*$.

Theorem 10. *The class of all maximal $(\tau, H_{d,k})$-bond-free subsets of $(\Sigma^k)^+$ is finite and equal to $\{S^{\oplus} \mid S \subseteq \Sigma^k$ and S is maximal satisfying $\tau(S) \cap H_d(S) = \emptyset\}$. In particular, if $d = 0$ then this class is equal to $\{S^{\oplus} \mid S \cup \tau(S) = \{v \in \Sigma^k \mid v \neq \tau(v)\},\ \tau(S) \cap S = \emptyset\}$.*

Note: *The above theorem holds also for subsets of $\Sigma^k\Sigma^*$ if we replace $S^{\oplus}$ with $S^{\otimes}$.*

According to Theorem 10, if K is a maximal subset of Σ^k satisfying $\tau(K) \cap H_d(K)$ then the language $K^{\oplus}$ is a maximal $(\tau, H_{d,k})$-bond-free subset of $(\Sigma^k)^+$. In the case of $d = 0$ the characterization of the maximal bond-free languages is quite explicit: Define any partition $\{S, \tau(S)\}$ of the set $\{v \in \Sigma^k \mid v \neq \tau(v)\}$ and then compute $S^{\oplus}$; this language will be maximal. The language L_2 considered in Example 3.1 is a particular instance of this type of construction [17].

The above theorem implies that every k-block $(\tau, H_{d,k})$-bond-free language L is included in a *regular* maximal such language. Statements of this type with L being regular have been obtained for various code-related properties and are of particular interest in the theory of codes [12]. In our case it is also interesting to note that the language L is not necessarily regular.

5 Discussion

We have considered the problem of undesirable bonds and proposed the property of (θ, sim)-bond-freedom for DNA languages, which addresses this problem when bonds between imperfect complements of DNA molecules are permitted. Using recent language theoretic tools, we were able to establish various decidability results about (θ, sim)-bond-freedom. The case where sim is the Hamming similarity has been considered by many authors. In this case, we have demonstrated interesting connections between our property and those of other authors, and have identified general construction *methods*. In particular, we have identified all DNA languages that are maximal with respect to the new property. This result is also applicable to the case of the θ-k-code property of [10]. Directions for future research include the following: (i) Derive a methodology for defining properties of DNA languages that would be able to address the uniqueness problem – called positive design problem in [18] – as independently of the application as possible. (ii) Elaborate on the proposed construction methods to obtain concrete constructions of languages that, in addition to being bond-free, they satisfy additional properties such as uniqueness and fixed GC-ratio. (iii) Explore further the subword closure operation from a theoretical at least point of view.

Acknowledgements

Research was partially supported by the Canada Research Chair Grant to L.K., NSERC Discovery Grants R2824A01 to L.K. and R220259 to S.K., and by the Grant Agency of Czech Republic, Grant 201/02/P079 to P.S.

References

1. Andronescu, M., Dees, D., Slaybaugh, L., Zhao, Y., Condon, A., Cohen, B., Skiena, S.: Algorithms for testing that sets of DNA words concatenate without secondary structure. In: [7], 182-195
2. Arita, M., Kobayashi, S.: DNA sequence design using templates. New Generation Computing **20** (2002) 263–277
3. Baum, E.: DNA sequences useful for computation. Pre-proceed. 2nd DIMACS Workshop on DNA-based computers, (1996) 122–127
4. Chen, J., Reif, J. (eds): Pre-proceed. 9th International Workshop on DNA-Based Computers, Madison, Wisconsin, 2003. Lecture Notes in Computer Science, Vol. 2943. Springer-Verlag, Berlin Heidelberg New York (2004)
5. Condon, A.E., Corn, R.M., Marathe, A.: On combinatorial DNA word design. Journal of Computational Biology, **8**:3 (2001) 201–220
6. Feldkamp, U., Saghafi, S., Rauhe, H.: DNASequenceGenerator - A program for the construction of DNA sequences. In: [11], 179–189
7. Hagiya, M., Ohuchi, A. (eds): Pre-proceed. 8th International Workshop on DNA-Based Computers, Saporo, Japan, 2002. Lecture Notes in Computer Science, Vol. 2568. Springer-Verlag, Berlin Heidelberg New York (2003)
8. Head, T.: Relativised code concepts and multi-tube DNA dictionaries. In: Calude, C.S., Păun, G. (eds.): Finite Versus Infinite: Contributions to an Eternal Dilemma. Springer-Verlag, London (2000) 175–186
9. Hussini, S., Kari, L., Konstantinidis, S.: Coding properties of DNA languages. In: [11], 107-118
10. Jonoska, N., Mahalingam, K.: Languages of DNA based code words. In: [4], 58–68
11. Jonoska, N., Seeman, N.C. (eds): Pre-proceed. 7th International Workshop on DNA-Based Computers, Tampa, Florida, 2001. Lecture Notes in Computer Science, Vol. 2340. Springer-Verlag, Berlin Heidelberg New York (2002)
12. Jürgensen, H., Konstantinidis, S.: Codes. In: G. Rozenberg, A. Salomaa (eds): Handbook of Formal Languages, Vol. I. Springer-Verlag, Berlin Heidelberg New York (1997) 511–607
13. Kari, L., Kitto, R., Thierrin, G.: Codes, involutions and DNA encoding. In: Brauer, W., Ehrig, H., Karhumäki, J., Salomaa, A. (eds.): Lecture Notes in Computer Science, Vol. 2300. Springer-Verlag, Berlin Heidelberg New York (2002) 376–393
14. Kari, L., Konstantinidis, S.: Language equations, maximality and error detection. Journal of Computer and System Sciences (to appear)
15. Kari, L., Konstantinidis, S., Losseva, E., Wozniak, G.: Sticky-free and overhang-free DNA languages. Acta Informatica **40** (2003) 119–157
16. Kari, L., Konstantinidis, S., Sosík, P.: On properties of bond-free DNA languages. Tech. report 609, Dept. Computer Science, Univ. Western Ontario, Canada (2003)
17. Kari, L., Konstantinidis, S., Sosík, P.: Bond-free languages: formalizations, maximality and construction methods. Tech. report 2004-01, Dept. Mathematics and Computing Science, Saint Mary's University, Halifax, Canada (2004). Electronic form available at `http://www.stmarys.ca/academic/science/compsci/`
18. Mauri, G., Ferretti, C.: Word Design for Molecular Computing: A Survey. In: [4], 37–46
19. Reif, J.H., LaBean, T.H., Pirrung, M., Rana, V.S., Guo, B., Kingsford, C., Wickham, G.S.: Experimental construction of very large scale DNA databases with associative search capability. In: [11], 231–247

20. Tanaka, F., Nakatsugawa, M., Yamamoto, M., Shiba, T., Ohuchi, A.: Developing support system for sequence design in DNA computing. In: [11], 129–137.
21. Tulpan, D.C., Hoos, H.H., Condon, A.E.: Stochastic Local Search Algorithms for DNA Word Design. In: [7], 229–241.

Preventing Undesirable Bonds Between DNA Codewords

Lila Kari[1], Stavros Konstantinidis[2], and Petr Sosík[1,3,⋆]

[1] Department of Computer Science, The University of Western Ontario, London, ON, Canada, N6A 5B7
{lila, sosik}@csd.uwo.ca

[2] Dept. of Mathematics and Computing Science, Saint Mary's University, Halifax, Nova Scotia, B3H 3C3 Canada
s.konstantinidis@stmarys.ca

[3] Institute of Computer Science, Silesian University, Opava, Czech Republic

Abstract. The input data for DNA computing must be encoded into the form of single or double DNA strands. As complementary parts of single strands can bind together forming a double-stranded DNA sequence, one has to impose restrictions on these sets of DNA words (=languages) to prevent them from interacting in undesirable ways. We recall a list of known properties of DNA languages which are free of certain types of undesirable bonds. Then we introduce a general framework in which we can characterize each of these properties by a solution of a uniform formal language inequation. This characterization allows us among others to construct (i) a uniform algorithm deciding in polynomial time whether a given DNA language possesses any of the studied properties, and (ii) in many cases also an algorithm deciding whether a given DNA language is maximal with respect to the desired property.

1 Introduction

A principle of DNA computing, in a nutshell, is to encode a task into a set of input DNA molecules so that their mutual (and highly parallel) reactions serve as a computational process, and produce a set of output molecules representing a *result* of the computing. Fundamental techniques are the operations of *hybridization (annealing)* and *denaturation (melting)* [16]. Given this framework, [14] and others distinguish two elementary subproblems of the design of sets of molecules for DNA experiments and computing:

– *Positive* design problem: to construct an input set of DNA molecules such that there exists a sequence of reactions to produce a desired final set – the result of the experiment.

⋆ Corresponding author.

– *Negative* design problem: the input set of molecules must not give way to the reactions that produce undesired molecules encoding a false result or blocking the desired reactions.

The negative design problem can be solved on a general basis by construction of a large set of molecules which do not allow undesired mutual reactions. Various subsets of this set are then used for concrete experiments. See e.g. [1, 3, 4, 5, 6, 8, 10, 13] for studies of properties of such sets (called also DNA languages) and methods of their construction.

In this paper we introduce the concept of general *(strictly) bond-free DNA language property*, and show that many of the previously studied properties are its special cases. Moreover, we construct a general quadratic-time algorithm deciding whether a given regular set of codewords satisfies any of these special cases of the bond-free property. We note that by the term *algorithm* we always mean a *deterministic* procedure, even if its input might be a nondeterministic formal automaton.

By utilizing and improving recent results in [9] on language inequations, we furthermore show that for many of the bond-free properties the *maximality* problem is decidable. Polynomial-time algorithms are presented for the case of θ-compliant finite sets and θ-non-overlapping regular sets. The proofs of these results are mostly nontrivial and due to page limitations can be found in [11].

2 Undesirable Bonds in DNA Languages

We represent the single-stranded DNA molecules by strings over the *DNA alphabet* $\Delta = \{A, C, T, G\}$. Generally, an *alphabet* Σ is a finite and nonempty set of symbols. The set of all words (over Σ) is denoted by Σ^*, including the *empty word* λ. The length of a word w is denoted by $|w|$. For an $n \geq 0$, w^n denotes the n concatenated copies of w. We denote the mirror image of the word w by w^R.

A language L is a set of words, i.e. a subset of Σ^*. For $n \geq 0$, we denote by L^n the set of all words $w_1 \cdots w_n$ such that each w_i is in L. We also write $L^* = L^0 \cup L^1 \cup L^2 \cup \cdots$, and L^+ for $L^* - \{\lambda\}$. The complement of L is $L^c = \Sigma^* - L$. The mirror image of L is $L^R = \{w^R \mid w \in L\}$.

A nondeterministic finite automaton (*NFA*) is a quintuple $A = (S, \Sigma, s_0, F, P)$ such that S is the finite and nonempty set of states, s_0 is the start state, F is the set of final states, and P is the set of productions of the form $sx \to t$, for $s, t \in S$, $x \in \Sigma$. If for every two productions $sx_1 \to t_1$ and $sx_2 \to t_2$ of an NFA we have that $x_1 \neq x_2$ then the automaton is called a *DFA* (deterministic finite automaton). The language accepted by the automaton A is denoted by $L(A)$. The *size* $|A|$ of the automaton A is the number $|S| + |P|$.

A mapping $\alpha : \Sigma^* \to \Sigma^*$ is called a *morphism* (*anti-morphism)* of Σ^* if $\alpha(uv) = \alpha(u)\alpha(v)$ (respectively $\alpha(uv) = \alpha(v)\alpha(u)$) for all $u, v \in \Sigma^*$.

An *involution* $\theta : \Sigma \to \Sigma$ of Σ is a mapping such that $\theta(\theta(x)) = x$ for all $x \in \Sigma$. It follows then that an involution θ is bijective and $\theta = \theta^{-1}$. An involution of Σ can be extended to either a morphism or an antimorphism of

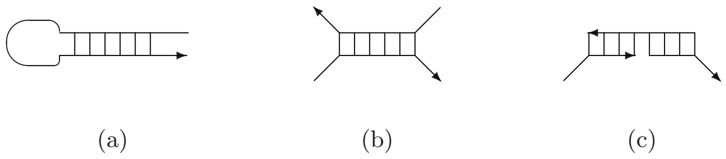


Fig. 1. Types of intramolecular (a) and intermolecular (b), (c) hybridizations

Σ^*. For example, if we extend the identity of Σ to an antimorphism of Σ^* we obtain the mirror-image involution of Σ^* that maps each word u into u^R.

If we consider the DNA-alphabet Δ, then the mapping $\tau : \Delta \to \Delta$ defined by $\tau(A) = T, \tau(T) = A, \tau(C) = G, \tau(G) = C$ can be extended in the usual way to an antimorphism of Δ^*. Then single strands $w_1, w_2 \in \Delta^*$ are complementary iff $w_1 = \tau(w_2)$. We refer the reader to [17] for further details on automata and formal languages.

Two types of unwanted hybridization are usually considered: *intramolecular*, Fig. 1 (a), and *intermolecular*, (b) and (c). The following properties of a DNA language $L \subseteq \Sigma^+$ preventing such a hybridization have been defined in [4, 8, 10].

(A) **θ-non-overlapping**: $L \cap \theta(L) = \emptyset$.

(B) **θ-compliant**: $\forall w \in L,\ x, y \in \Sigma^*,\ w, x\theta(w)y \in L \Rightarrow xy = \lambda$.

(C) **θ-p-compliant**: $\forall w \in L,\ y \in \Sigma^*,\ w, \theta(w)y \in L \Rightarrow y = \lambda$.

(D) **θ-s-compliant**: $\forall w \in L,\ y \in \Sigma^*,\ w, y\theta(w) \in L \Rightarrow y = \lambda$.

(E) **strictly θ-compliant**: both θ-compliant and θ-non-overlapping.

(F) **θ-free**: $L^2 \cap \Sigma^+\theta(L)\Sigma^+ = \emptyset$.

(G) **θ-sticky-free**: $\forall w \in \Sigma^+,\ x, y \in \Sigma^*,\ wx, y\theta(w) \in L \Rightarrow xy = \lambda$.

(H) **θ-3′-overhang-free**: $\forall w \in \Sigma^+,\ x, y \in \Sigma^*,\ wx, \theta(w)y \in L \Rightarrow xy = \lambda$.

(I) **θ-5′-overhang-free**: $\forall w \in \Sigma^+,\ x, y \in \Sigma^*,\ xw, y\theta(w) \in L \Rightarrow xy = \lambda$.

(J) **θ-overhang-free**: both θ-3′-overhang-free and θ-5′-overhang-free.

We agree to say that a language L containing the empty word has one of the above properties if $L \setminus \{\lambda\}$ has that property. Observe that (F) corresponds to Fig. 1 (c), while others are special cases of (b).

In [6], a θ-non-overlapping language is called to be *strictly* θ. Generally, if any other property holds in conjunction with (A), we add the qualifier *strictly.* Further properties have been defined in [6]. We denote by $\mathrm{Sub}_k(L)$ the set of all subwords of L of the length k. The following property corresponds to Fig. 1 (a):

(K) **$\theta(k, m_1, m_2)$-subword compliant**: $\forall u \in \Sigma^k,\ \Sigma^* u \Sigma^m \theta(u) \Sigma^* \cap L = \emptyset$ for $k > 0,\ m_1 \le m \le m_2$.

(L) **θ-k-code**: $\mathrm{Sub}_k(L) \cap \mathrm{Sub}_k(\theta(L)) = \emptyset,\ k > 0$.

The following property is defined for $\theta = I$, the identity relation, in [3]. A language L is called

(M) **solid** if:
1. $\forall x, y, u \in \Sigma^*$, $u, xuy \in L \Rightarrow xy = \lambda$, and
2. $\forall x, y \in \Sigma^*, u \in \Sigma^+$, $xu, uy \in L \Rightarrow xy = \lambda$.

L is *solid relative to an* $M \subseteq \Sigma^*$ if 1. and 2. above hold only for $w = pxuyq \in M$. L is called *comma-free* if it is solid relative to L^*. Solid languages are also used in [10] as a tool for constructing error-detecting DNA languages that are invariant under bio-operations. We refer to [6, 10, 11] for further examples and for mutual relations between classes of a DNA languages satisfying these properties.

3 Binary Word Operations

Binary word operations on are extensively used in the following sections for representing interaction of DNA molecules. A *binary word operation* is a mapping $\diamondsuit : \Sigma^* \times \Sigma^* \to 2^{\Sigma^*}$, where 2^{Σ^*} is the set of all subsets of Σ^*. The notion can be extended to languages X and Y as follows:

$$X \diamondsuit Y = \bigcup_{u \in X, v \in Y} u \diamondsuit v. \tag{1}$$

The left and the right inverse $\diamondsuit^l$ and $\diamondsuit^r$ of $\diamondsuit$, respectively, are defined as

$$w \in (x \diamondsuit v) \text{ iff } x \in (w \diamondsuit^l v) \text{ iff } v \in (x \diamondsuit^r w), \text{ for all } v, x, w \in \Sigma^*.$$

Examples of binary word operations are catenation, quotient, insertion, deletion, shuffle etc. See [7, 9, 15] for more details.

Further we introduce word operations on trajectories [2, 12, 15]. Consider a *trajectory alphabet* $V = \{0, 1\}$ and assume $V \cap \Sigma = \emptyset$. We call *trajectory* any string $t \in V^*$. A trajectory is essentially a syntactical condition which specifies how an operation $\diamondsuit$ is applied to the letters of its two operands. Let $t \in V^*$ be a trajectory and let α, β be two words over Σ.

Definition 1. *The shuffle of α with β on the trajectory t, denoted by $\alpha \sqcup\!\sqcup_t \beta$, is defined as follows:*

$$\alpha \sqcup\!\sqcup_t \beta = \{\alpha_1\beta_1 \ldots \alpha_k\beta_k \mid \alpha = \alpha_1 \ldots \alpha_k,\ \beta = \beta_1 \ldots \beta_k,\ t = 0^{i_1}1^{j_1} \ldots 0^{i_k}1^{j_k},$$
$$\text{where } |\alpha_m| = i_m \text{ and } |\beta_m| = j_m \text{ for all } m,\ 1 \le m \le k\}.$$

Example 2. Let $\alpha = a_1a_2 \ldots a_8$, $\beta = b_1b_2 \ldots b_5$, $t = 0^31^20^310101$. The shuffle of α and β on the trajectory t is: $\alpha \sqcup\!\sqcup_t \beta = \{a_1a_2a_3b_1b_2a_4a_5a_6b_3a_7b_4a_8b_5\}$.

Definition 3. *The deletion of β from α on the trajectory t is the following binary word operation:*

$$\alpha \rightsquigarrow_t \beta = \{\alpha_1 \ldots \alpha_k \mid \alpha = \alpha_1\beta_1 \ldots \alpha_k\beta_k,\ \beta = \beta_1 \ldots \beta_k,\ t = 0^{i_1}1^{j_1} \ldots 0^{i_k}1^{j_k},$$
$$\text{where } |\alpha_m| = i_m \text{ and } |\beta_m| = j_m \text{ for all } m,\ 1 \le m \le k\}.$$

Example 4. Let $\alpha = babaab$, $\beta = bb$ and assume that $t = 001001$. The deletion of β from α on the trajectory t is: $\alpha \rightsquigarrow_t \beta = \{baaa\}$.

The operations can be extended to sets of trajectories as follows: $\alpha \diamondsuit_T \beta = \bigcup_{t \in T} \alpha \diamondsuit_t \beta$, where $\diamondsuit$ stands for $\sqcup\!\sqcup$ or $\rightsquigarrow$, respectively. The operations $\sqcup\!\sqcup_T$ and $\rightsquigarrow_T$ generalize further to languages due to (1).

4 Bond-Free DNA Languages

The notion of DNA language property from Section 2 can be formalized as follows: a property $\mathcal{P}$ is a mapping $\mathcal{P} : 2^{\Sigma^*} \longrightarrow \{\text{true}, \text{false}\}$. We say that a language $L \subseteq \Sigma^*$ has (or satisfies) the property $\mathcal{P}$ if $\mathcal{P}(L) = \text{true}$.

Definition 5. *A language property $\mathcal{P}$ is called a* bond-free property of degree 2 *if there exist binary word operations $\diamondsuit_{\text{lo}}, \diamondsuit_{\text{up}}$ such that for an arbitrary $L \subseteq \Sigma^*$, $\mathcal{P}(L) = \text{true}$ iff*

$$\forall w \in \Sigma^+,\ x, y \in \Sigma^*, (w \diamondsuit_{\text{lo}} x \cap L \neq \emptyset,\ w \diamondsuit_{\text{up}} y \cap \theta(L) \neq \emptyset) \Rightarrow\ xy = \lambda. \quad (2)$$

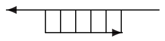
The phrase *degree 2* is used to stress the fact that the property describes bonds of *two* single DNA strands. In the remainder of this paper we write simply *bond-free property* for a bond-free property of degree two.

Furthermore, in this and the following section we assume that $\diamondsuit_{\text{lo}} = \sqcup\!\sqcup_{T_{\text{lo}}}$ and $\diamondsuit_{\text{up}} = \sqcup\!\sqcup_{T_{\text{up}}}$ for some trajectory sets $T_{\text{lo}}, T_{\text{up}} \subseteq V^*$.

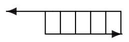
Theorem 6. *The language properties (B), (C), (D), (G), (H), (I), (M.1), (M.2) are bond-free properties. Moreover, the associated sets of trajectories $T_{\text{lo}}, T_{\text{up}}$ are regular.*

Proof. Assume that θ is an antimorphism and define the sets of trajectories T_{lo}, T_{up} as follows.

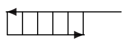
(B) θ-**compliant**: $T_{\text{lo}} = 0^+$, $T_{\text{up}} = 1^*0^+1^*$.



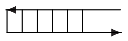
(C) θ-p-**compliant**: $T_{\text{lo}} = 0^+$, $T_{\text{up}} = 1^*0^+$.



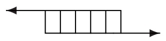
(D) θ-s-**compliant**: $T_{\text{lo}} = 0^+$, $T_{\text{up}} = 0^+1^*$.



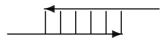
(G) θ-**sticky-free**: $T_{\text{lo}} = 0^+1^*$, $T_{\text{up}} = 0^+1^*$.



(H) θ-**3′-overhang-free**: $T_{\text{lo}} = 0^+1^*$, $T_{\text{up}} = 1^*0^+$.



(I) θ-**5′-overhang-free**: $T_{\text{lo}} = 1^*0^+$, $T_{\text{up}} = 0^+1^*$.



Consider e.g. the property (H), θ-3'-overhang-freedom. Then $w \sqcup\!\sqcup_{T_{\text{lo}}} x = \{wx\}$ and $w \sqcup\!\sqcup_{T_{\text{up}}} y = \{yw\}$. The relations in (2) adopt the form $wx \in L$, $yw \in \theta(L)$. This is equivalent to $wx \in L$, $\theta(w)\theta(y) \in L$. As $xy = \lambda$ iff $x\theta(y) = \lambda$,

(2) corresponds to the definition of (H) in Section 2. The proofs of the other mentioned properties are analogous.

If θ is a morphism, then all the sets of trajectories $T_{\rm up}$ must be replaced by the reversed sets $T_{\rm up}^R$, the proof technique remaining unchanged. $\square$

Observe that $T_{\rm lo}, T_{\rm up}$ for a certain property corresponds to the "shape" of the bonds prohibited in languages satisfying the property, see attached figures.

The main reason for introducing Definition 5 is the characterization of bond-free properties via language inequations.

Theorem 7. *For each bond-free property $\mathcal{P}$ there are regular sets of trajectories T_1, T_2 and a binary word operation $\boxminus_{\mathcal{P}}$ defined as*

$$x \boxminus_{\mathcal{P}} y = ((x \sqcup\!\sqcup_{(01)^*} T_{\rm lo}) \sqcup\!\sqcup_{T_1} (\theta(y) \sqcup\!\sqcup_{(01)^*} T_{\rm up})) \rightsquigarrow_{T_2} K_1, \tag{3}$$

such that $\mathcal{P}(L) = true$ for an $L \subseteq \Sigma^*$ iff $L \boxminus_{\mathcal{P}} L \subseteq K_2$.

This characterization allows us to answer decidability questions "Is $\mathcal{P}(L) =$ true for a given language L and a bond-free property $\mathcal{P}$?"

Theorem 8. *Let $\mathcal{P}$ be a bond-free property associated with regular sets of trajectories $T_{\rm lo}, T_{\rm up}$. The following problem is decidable in quadratic time w.r.t $|A|$:*
Input: an NFA A.
Output: Y/N depending on whether $L(A)$ satisfies $\mathcal{P}$.

Proof. Can be found in [11] using known results about word operations on trajectories in [2, 12, 15]. $\square$

In [4] the decidability of the properties (D) and (F) for regular sets of words was shown. In [3] the decidability of (M) in quadratic time is proven. In [5] an algorithm deciding (F) in quadratic time for finite sets of codewords is presented. The following corollary extends and generalizes these previous results for the case of regular sets of DNA codewords.

Corollary 9. *The following problem is decidable in quadratic time w.r.t. $|A|$:*
Input: an NFA A.
Output: Y/N depending on whether $L(A)$ satisfies any of the properties (B), (C), (D), (G), (H), (I), (J) (M).

On the other hand, it is known [4] that for some bond-free properties, e.g. (B) and (F), there is no such algorithm in the case of context-free DNA languages.

5 Maximal Bond-Free Languages

The approach used in the previous section can be applied also to maximality problems ("Is $L \subseteq M$ maximal w.r.t. a bond-free property $\mathcal{P}$?"). The symbol $M \subseteq \Sigma^*$ represents the set of all applicable/constructible DNA strands in a case at hand. The following theorem is based on nontrivial results concerning language inequations in [9].

Theorem 10. *Let $\mathcal{P}$ be a bond-free property and $M \subseteq \Sigma^+$ a set of words. For a language $L \subseteq M$ satisfying $\mathcal{P}$, denote*

$$R = M - (L \cup L \boxminus^r_{\mathcal{P}} K_2^c \cup K_2^c \boxminus^l_{\mathcal{P}} L), \tag{4}$$
$$Q = \{z \in \Sigma^* \mid z \boxminus_{\mathcal{P}} z \cap K_2^c \neq \emptyset\}, \tag{5}$$

where $\boxminus_{\mathcal{P}}$ is defined by (3) and $K_2 = (\Sigma \cup V)^* 0 (\Sigma \cup V)^* \cup \{\lambda\}$. Then L is a maximal subset of M satisfying $\mathcal{P}$ iff $R - Q = \emptyset$.

After calculating the inverses of the operation $\boxminus_{\mathcal{P}}$, we obtain a decision algorithm concerning maximal bond-free languages.

Theorem 11. *Consider a fixed involution θ. Let $\mathcal{P}$ be one of the properties (B), (C), (D), (G) if θ is an antimorphism, and one of (B), (C), (D), (H), (I) if θ is a morphism.*

Let $M \subseteq \Sigma^+$ be a regular set of words, and $L \subseteq M$ a regular language satisfying $\mathcal{P}$. Then there is an algorithm deciding whether L is a maximal subset of M satisfying $\mathcal{P}$.

Let A be a DFA accepting L. The algorithms described in the above theorem may require an exponential number of steps w.r.t. $|A|$ in the worst case. But one can obtain a polynomial-time algorithm at least for finite languages.

Theorem 12. *The following problem is decidable in time $O(\|L\|^3|A|)$, where $\|L\|$ is the quantity $\sum_{w \in L} |w|$.*

Input: DFA A and a finite language L such that $L \subseteq L(A)$ and L satisfies the property (B).

Output: Y/N, depending on whether L is a maximal subset of $L(A)$ satisfying (B).

The language inequation approach can be used also for characterization of supersets of non-maximal bond-free DNA languages without breaking the given bond-free property, see [11] for details. However, to construct such a superset may require an exponential number of steps w.r.t. $|A|$.

6 Strictly Bond-Free Languages

In this section we focus mostly on the strict versions of the DNA language properties (B)–(L), i.e. their conjunctions with (A). As one can easily observe, the property (E) is equal to strictly (B), hence we do not refer to (E) in the sequel. The following concept of a *strictly bond-free* property generalizes these properties. However, (non-strictly) (L) is also a special case of the strictly bond-free property.

Definition 13. *A language property $\mathcal{P}$ is called the* strictly bond-free property of degree 2 *if there are binary word operations $\diamondsuit_{\mathrm{lo}}$, $\diamondsuit_{\mathrm{up}}$ and an involution θ such that for an arbitrary $L \subseteq \Sigma^*$, $\mathcal{P}(L) = true$ iff*

$$\forall w, x, y \in \Sigma^* \ (w \diamondsuit_{\mathrm{lo}} x \cap L \neq \emptyset,\ w \diamondsuit_{\mathrm{up}} y \cap \theta(L) \neq \emptyset) \Rightarrow w = \lambda. \tag{6}$$

Again, in the remainder of this paper we write simply *strictly bond-free property* for the strictly bond-free property of degree two.

Theorem 14. *The language properties (A), strictly (B)–(D), strictly (G)–(I), (L), strictly (L) are strictly bond-free properties.*

Proof. Let $\diamondsuit_{\mathrm{lo}} = \sqcup\!\sqcup_{T_{\mathrm{lo}}}$ and $\diamondsuit_{\mathrm{up}} = \sqcup\!\sqcup_{T_{\mathrm{up}}}$, where T_{lo} and T_{up} are the sets of trajectories used in the proof of Theorem 6. The rest of the proof relies on similar techniques. □

Similarly as in Theorem 7, one can characterize strictly bond-free properties via language (in)equations.

Theorem 15. *For each strictly bond-free property $\mathcal{P}$ there is a binary word operation $\boxdot_{\mathcal{P}}$ defined as*

$$x \boxdot_{\mathcal{P}} y = (x \diamondsuit^{l}_{\mathrm{lo}} \Sigma^*) \rightsquigarrow_{1^+} (\theta(y) \diamondsuit^{l}_{\mathrm{up}} \Sigma^*), \tag{7}$$

such that for a language $L \subseteq \Sigma^*$, $\mathcal{P}(L) = true$ iff $L \boxdot_{\mathcal{P}} L = \emptyset$.

As a consequence one can derive a quadratic time decision algorithm for regular DNA languages.

Theorem 16. *Let $\mathcal{P}$ be a strictly bond-free property associated with operations $\diamondsuit_{\mathrm{lo}} = \sqcup\!\sqcup_{T_{\mathrm{lo}}}$, $\diamondsuit_{\mathrm{up}} = \sqcup\!\sqcup_{T_{\mathrm{up}}}$, with regular sets of trajectories $T_{\mathrm{lo}}, T_{\mathrm{up}}$. Then the following problem is decidable in quadratic time w.r.t. $|A|$:*
Input: an NFA A.
Output: Y/N depending on whether $L(A)$ satisfies $\mathcal{P}$.

Corollary 17. *Let $\mathcal{P}$ be any of the properties (A), strictly (B) – strictly (D), strictly (G) – strictly (J), (L), strictly (L). The following problem is decidable in quadratic time w.r.t. $|A|$:*
Input: an NFA A.
Output: Y/N depending on whether $L(A)$ satisfies $\mathcal{P}$.

On the other hand, one can easily show [11] that e.g. for the property (A), there is no decision algorithm in the context-free case.

7 Maximal Strictly Bond-Free Languages

We focus on maximality problems (see section 5) w.r.t. a strictly bond-free property $\mathcal{P}$. For the case of the θ-non-overlapping regular languages, the problem is decidable in polynomial time, for some other properties the polynomial-time algorithm for regular languages is not known.

Theorem 18. *The following problem is decidable in time $O((|A| \cdot |A_\theta| \cdot |A_M|)^3)$.*
Input: DFA's A, A_θ and an NFA A_M such that $L(A) = \theta(L(A_\theta)) \subseteq L(A_M)$ and $L(A)$ is θ-non-overlapping.
Output: Y/N, depending on whether $L(A)$ is a maximal θ-non-overlapping subset of $L(A_M)$.

Theorem 19. *Consider a fixed involution θ. Let $\mathcal{P}$ be any of the properties strictly (B) – strictly (D), strictly (G), (L), strictly (L) if θ is an antimorphism. Let $\mathcal{P}$ be any of strictly (B) – strictly (D), strictly (H), strictly (I), (L), strictly (L) if θ is a morphism.*

Let $M \subseteq \Sigma^+$ be a regular set of words and $L \subseteq M$ a regular language satisfying $\mathcal{P}$. Then there is an algorithm deciding whether L is a maximal subset of M satisfying $\mathcal{P}$.

Similarly as in Section 5, supersets of non-maximal languages satisfying a certain strictly bond-free property can be also characterized.

8 Summary

We proposed a sequence of algorithms solving decision problems of DNA languages without undesirable bonds. The results are summarized in Tables 1 and 2. The abbreviations *REG* and *CF* denote the classes of regular and context-free languages, respectively. In the column θ, the symbol A denotes antimorphism and M denotes morphism, $*$ stands for an arbitrary involution. In the columns corresponding to particular properties (B)–(M), D stands for decidable, Q for quadratic-time decidable, P for polynomial-time decidable, U for undecidable and ? for an open problem.

Furthermore we presented a polynomial-time algorithm deciding maximality of a *finite* DNA language w.r.t. the property (B).

Among major open questions we mention the study of fast algorithms for construction of finite bond-free languages, methods preventing imperfect bonds (with bulges, non-complementary pairs etc.) between DNA strands, and study of influence of the secondary DNA structure and free energy of single strands.

Table 1. Decision problems of non-strict DNA language properties

Problem	**Class**	θ	**Properties** (B)	**(C)**	**(D)**	**(G)**	**(H)**	**(I)**	**(J)**	**(L)**	**(M)**
Does a given language satisfy the property $\mathcal{P}$?	*REG*	$*$	Q	Q	Q	Q	Q	Q	Q	Q	Q
	CF	$*$	U	?	?	?	?	?	?	?	?
Is a given language maximal w.r.t. $\mathcal{P}$?	*REG*	A	D	D	D	D	?	?	?	D	-
	REG	M	D	D	D	?	D	D	?	D	?

Table 2. Decision problems of strict DNA language properties

Problem	**Class**	θ	**Properties** (A)	**(B)**	**(C)**	**(D)**	**(G)**	**(H)**	**(I)**	**(J)**	**(L)**
Does a given language satisfy the property $\mathcal{P}$?	*REG*	$*$	Q	Q	Q	Q	Q	Q	Q	Q	Q
	CF	$*$	U	?	?	?	?	?	?	?	?
Is a given language maximal w.r.t. $\mathcal{P}$?	*REG*	A	P	D	D	D	D	?	?	?	D
	REG	M	P	D	D	D	?	D	D	?	D

Acknowledgements

Research was partially supported by the Canada Research Chair Grant to L.K., NSERC Discovery Grants R2824A01 to L.K. and R220259 to S.K., and by the Grant Agency of Czech Republic, Grant 201/02/P079 to P.S.

References

1. M. Arita, S. Kobayashi, DNA sequence design using templates. *New Generation Computing* **20** (2002), 263–277.
2. M. Domaratzki, *Deletion Along Trajectories.* Tech. Report 464-2003, School of Computing, Queen's University, 2003, and submitted for publication.
3. T. Head, Relativised code concepts and multi-tube DNA dictionaries. In C.S. Calude, G. Păun, *Finite Versus Infinite: Contributions to an Eternal Dilemma*, Springer-Verlag, London, 2000, 175–186.
4. S. Hussini, L. Kari, S. Konstantinidis, Coding properties of DNA languages. *theoretical Computer Science* **290/3** (2002), 1557-1579.
5. N. Jonoska, D. Kephart, K. Mahalingam, Generating DNA code words. *Congressus Numerantium* **156** (2002), 99–110.
6. N. Jonoska, K. Mahalingam, Languages of DNA based code words. In J. Chen, J. Reif (Eds.), Preproceedings of DNA9, June 1–4, 2003, Madison, Wisconsin, pp. 58–68.
7. L. Kari, On insertion and deletion in formal languages, *PhD thesis*, University of Turku, Finland, 1991.
8. L. Kari, R. Kitto, G. Thierrin, Codes, involutions and DNA encoding. In W. Brauer, H. Ehrig, J. Karhumäki, A. Salomaa (Eds.), *Lecture Notes in Computer Science* **2300** (2002), 376–393.
9. L. Kari, S. Konstantinidis, Language equations, maximality and error detection. Submitted for publication.
10. L. Kari, S. Konstantinidis, E. Losseva, G. Wozniak, Sticky-free and overhang-free DNA languages. *Acta Informatica* **40** (2003), 119–157.
11. L. Kari, S. Konstantinidis, P. Sosík, *On Properties of Bond-Free DNA Languages.* Dept. of Computer Science Tech. Report No. 609, Univ. of Western Ontario, 2003, and submitted for publication.
12. L. Kari, P. Sosík, *Language deletion on trajectories.* Dept. of Computer Science Technical Report No. 606, University of Western Ontario, London, 2003.
13. A. Marathe, A.E. Condon, R.M. Corn, On combinatorial DNA words design. *J. Computational Biology*, **8**:3, 2001.
14. G. Mauri, C. Ferretti. Word Design for Molecular Computing: A Survey. In J. Chen and J.H. Reif (Eds.), *DNA Computing, 9th International Workshop on DNA Based Computers*, *Lecture Notes in Computer Science* **2943** (2004), 37–46.
15. A. Mateescu, G. Rozenberg, A. Salomaa, Shuffle on trajectories: syntactic constraints, TUCS technical report No. 41, Turku Centre for Computer Science, 1996, and *Theoretical Computer Science* **197** (1998), 1–56.
16. G. Păun, G. Rozenberg, A. Salomaa, *DNA Computing. New Computing Paradigms,* Springer-Verlag, Berlin, 1998.
17. G. Rozenberg, A. Salomaa (Eds.), *Handbook of Formal Languages,* Springer-Verlag, Berlin, 1997.

Testing Structure Freeness of Regular Sets of Biomolecular Sequences[*] (Extended Abstract)

Satoshi Kobayashi

Dept. of Computer Science,
Univ. of Electro-Communications
satoshi@cs.uec.ac.jp

Abstract. $p \xrightarrow{x} q$ This paper discusses a problem of checking the structure freeness of DNA and RNA sequences. A set R of sequences over Σ is said to be *structure free* if for every $\alpha \in R$, the minimum free energy of α is greater than or equal to 0. It was open whether or not there exists a polynomial time algorithm for testing the structure freeness of a given regular set R. In this paper, we will solve this problem.

Keywords: Molecular Computing, DNA Computing, Sequence Design, Secondary Structures.

1 Introduction

Since the Adleman's pioneering work on the computation of directed Hamiltonian path problem using molecular biological experimental technique ([Adl94]), the design of DNA and RNA sequences is one of the most practical and important research topics in DNA computing ([BC01]). One of the most important requirements for the sequence design is that we should design sequences so that they do not form any secondary structure, in other words, they are *structure free*. In order to solve this design problem, we need to devise an efficient algorithm to test the structure freeness of sequences. In this paper, we will focus on this structure freeness test problem.

This paper discusses on an open question posed in [Con03] and [ADS02]. A regular set R of sequences over Σ is said to be *structure free* if for every $\alpha \in R$, the minimum free energy of α is greater than or equal to 0. The problem is to decide whether the given R is structure free or not. In this paper, we will give a polynomial time algorithm to solve this problem.

[*] This work is supported in part by Grant-in-Aid for Scientific Research on Priority Area No.14085205, Ministry of Education, Culture, Sports, Science and Technology, Japan. It was also supported in part by Grant-in-Aid for Exploratory Research NO.13878054, Japan Society for the Promotion of Science.

2 Preliminaries

Let Σ be an alphabet $\{A, C, G, T\}$ or $\{A, C, G, U\}$. A letter in Σ is also called a *base*. By ϵ, we denote an empty string over Σ. A DNA or RNA sequence can be seen as an oriented sequence of consecutively covalently bonded bases: one of its end is called *5' end*, and the other is called *3' end*. A string is regarded as a base sequence ordered from $5'$-end to $3'$-end direction. Consider a string α over Σ. By $|\alpha|$ we denote the length of α. For a finite set or multiset X, by $|X|$ we denote the number of elements of X. For integers i, j such that $1 \leq i \leq j \leq |\alpha|$, by $\alpha[i, j]$ we denote the substring of α starting from the ith letter and ending at the jth letter of α. In case of $i = j$, we write $\alpha[i]$. In case of $i > j$ or $|\alpha| = 0$, $\alpha[i, j]$ represents an empty string ϵ.

2.1 Secondary Structure

We will partly follow the terminologies and notations used in [SKM83]. Let us introduce a relation $\theta \subseteq \Sigma \times \Sigma$ defined by $\theta = \{ (A, T), (T, A), (G, C), (C, G), (G,T), (T, G)\}$, for representing Watson-Crick and non-Watson-Crick base pairs[1]. A hydrogen bond between the ith base and jth base of a string α can be formed only if $(\alpha[i], \alpha[j]) \in \theta$ holds. The hydrogen bond between the ith base and jth base is denoted by (i, j). The hydrogen bond (i, j) is also called a *base pair*. Without loss of generality, we may always assume that $i < j$ for a base pair (i, j). For two base pairs (i, j) and (k, l), we write $(i, j) \prec (k, l)$ if $i < k$ holds. A secondary structure of α is a finite set of such base pairs of α. A string α together with its secondary structure T is called a *structured string*, and written α (T). For representing the ith base in α (T), we often use the integer i.

In this paper, we consider only *pseudo-knot free* secondary structures T such that there exist no base pairs $(i, j), (k, l) \in T$ satisfying $i < k < j < l$. In the sequel, we assume that every secondary structure is pseudo-knot free.

For three bases i, j, and r in α (T), we say that i and j *surround* r if $i < r < j$. In case of $(i, j) \in T$, we also say that the base pair (i, j) *surrounds* r. For a base pair (p, q), we say that i and j *surround* (p, q) if i and j surround both p and q. In case of $(i, j) \in T$, we also say that (i, j) *surrounds* (p, q), written $(p, q) < (i, j)$ or $(i, j) > (p, q)$. A base pair (p, q) or an unpaired base r is said to be *accessible from* i and j, if it is surrounded by i and j and is not surrounded by any base pair (k, l) such that (k, l) is surrounded by i and j. In case of $(i, j) \in T$, we also say that it is accessible from a base pair (i, j).

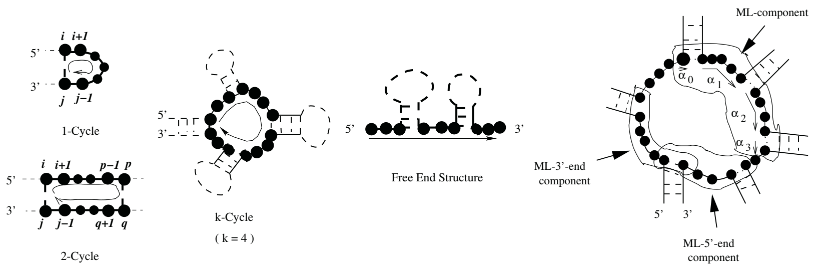
For each base pair $bp = (i, j) \in T$, we define a *cycle* $c(bp)$ as a substructure consisting of the base pair (i, j) together with any base pairs (p_1, q_1), (p_2, q_2), ... accessible from (i, j) and any unpaired bases accessible from (i, j). If $c(bp)$ contains k base pairs (including the base pair (i, j)), it is said to be a k-cycle or a *cycle of order* k. In case of $k = 1$, we often call it a *hairpin*. In case of $k \geq 3$, it is often called a *multiple loop*. In these definitions, the base pair (i, j) is called a *closing base pair* of the cycle. (See Figure 1 (a).)

[1] For the case of RNA strings, replace the letter T by U.

Furthermore, in case of $(1, |\alpha|) \notin T$, the substructure of α (T) consisting of the base pairs and the unpaired bases accessible from 1 and $|\alpha|$ together with the bases 1 and $|\alpha|$ is called a *free end structure* of α (T).

The *loop length* of a 1-cycle c with a base pair (i, j) is defined as the number of unpaired bases $j-i-1$. The *loop length* of a 2-cycle with base pairs (i, j) and (p, q) $((p, q) < (i, j))$ is also defined as the number of unpaired bases $p-i+j-q-2$. The *loop length mismatch* of a 2-cycle with base pairs (i, j) and (p, q) $((p, q) < (i, j))$ is defined as $|(p-i)-(j-q)|$.

By $\overline{\uparrow \alpha \downarrow}$ we denote a 1-cycle consisting of a string α with a base pair between $\alpha[1]$ and $\alpha[|\alpha|]$. By $\left|\begin{smallmatrix}\uparrow\alpha\\ \beta\downarrow\end{smallmatrix}\right|$ we denote a 2-cycle consisting of two strings α and β with base pairs between $\alpha[1]$ and $\beta[|\beta|]$ and between $\alpha[|\alpha|]$ and $\beta[1]$.



(a) Basic Secondary Structures (b) Components of a Multiple Loop

Fig. 1. Secondary Structure

2.2 Free Energy Calculation

Free energy value is assigned to each cycle or free end structure. Experimental evidence is used to determine such free energy values. The method for assigning free energy values is given as follows [2]: (See Figure 1 (a).)

1. The free energy $E(c)$ of a 1-cycle c with a base pair (i, j) is dependent on the base pair (i, j), the unpaired bases $i+1, j-1$ adjacent to (i, j), and its loop length l:

$$E(c) = f_1(\alpha[i], \alpha[j], \alpha[i+1], \alpha[j-1]) + g_1(l).$$

2. The free energy $E(c)$ of a 2-cycle c with base pairs (i, j) and (p, q) $((p, q) < (i, j))$ is dependent on the base pairs (i, j), (p, q), the unpaired bases $i+1, j-1$ adjacent to (i, j), the unpaired bases $p-1, q+1$ adjacent to (p, q), its loop length l, and its loop length mismatch d:

$$E(c) = f_2(\alpha[i], \alpha[j], \alpha[p], \alpha[q], \alpha[i+1], \alpha[j-1], \alpha[p-1], \alpha[q+1]) + g_2(l) + g_3(d),$$

[2] The calculation method presented here is in a general form so that it can be specialized to be equivalent to the one used in standard RNA packages such as ViennaRNA([HFS94]).

3. The free energy $E(c)$ of a $(k+1)$-cycle c $(k \geq 2)$ with a closing base pair (i,j) and the base pairs (p_1,q_1), (p_2,q_2), ..., (p_k,q_k) accessible from (i,j) is dependent on the base pairs (i,j), (p_l,q_l) $(l=1,...,k)$, the unpaired bases $i+1, j-1$ adjacent to (i,j), the unpaired bases p_l-1, q_l+1 adjacent to (p_l,q_l) $(l=1,...,k)$, the number n_b $(=k+1)$ of base pairs in c, and the number n_u of unpaired bases in c:

$$\begin{aligned} E(c) = {} & m_1(\alpha[i],\alpha[j],\alpha[i+1]) + m_2(\alpha[i],\alpha[j],\alpha[j-1]) + \\ & \Sigma_{l=1}^{k}(m_1(\alpha[q_l],\alpha[p_l],a[q_l+1]) + m_2(\alpha[q_l],\alpha[p_l],a[p_l-1])) + \\ & M_b * n_b + M_u * n_u + C_M, \end{aligned}$$

4. Although the free energy $E(c)$ of a free end structure c is defined in a similar way as that of k-cycle $(k \geq 3)$, we assume in this paper that it is defined as 0. This simplification is only because of space constraint. The result of this paper is easily extended to the case that $E(c)$ of a free end structure c is appropriately defined.

In these definitions, the functions f_1, g_1, f_2, g_2, g_3, m_1, m_2 are experimentally obtained functions. The constants M_b, M_u, C_M are also experimentally obtained. In the case that there does not exist an unpaired base which is adjacent to a given base pair, an empty string ϵ is used as an argument of those functions.

We assume that M_b, M_u, C_M are nonnegative constants. For each function g_i $(i=1,2,3)$, we assume that there exists a nonnegative integer L_i such that for the range $l > L_i$, $g_i(l)$ is weakly monotonically increasing. Furthermore, we assume that all the above functions are computable in constant time.

Let $c_1, ..., c_k$ be the cycles contained in α (T). Then, the free energy $E(\alpha\ (T))$ of α (T) is given by $E(\alpha\ (T)) = \Sigma_{i=1}^{k} E(c_i)$.

2.3 Structure Freeness Test Problem

In this paper, we will consider the problem of testing whether a given regular set of strings is structure free or not. The problem is formally defined in the following way.

Let R be a regular language over Σ. Then, we say that R is *structure free* if for any structured string α (T) such that $\alpha \in R$ and T is pseudo-knot free, it holds that $E(\alpha\ (T)) \geq 0$. We have interests in deciding for given R, whether or not R is structure free. In section 4, we will give a polynomial time algorithm for solving this problem.

For specifying a regular language R, we use a labeled directed graph with initial and final vertices. Let $M = (V, E, \sigma, I, F)$, where V is a finite set of vertices, E is a subset of $V \times V$, σ is a label function from V to $\Sigma \cup \{\epsilon\}$, and $I, F \subseteq V$. For $p, q \in V$ and $x \in \Sigma^*$, we write $p \xrightarrow{x} q$ if there is a path with labels x from p to q in M. Note that x contains the labels $\sigma(p)$ and $\sigma(q)$. We write $p \rightarrow q$ if $p \xrightarrow{x} q$ for some $x \in \Sigma^*$. A string α is *accepted* by M if $p \xrightarrow{\alpha} q$ for some $p \in I$ and $q \in F$. This graph representation could be regarded as a Moore type machine with no edge labels. Thus, the set $L(M)$ of strings accepted by M is regular. Furthermore, it is also clear that every regular language can be accepted by some graph M. A graph M is said to be *trimmed* if every vertex is reachable from an initial vertex and has a path to a final vertex.

For subsequent convenience, we introduce an array *Len* in the following way. For each pair of vertices $p, q \in V$ with $\sigma(p), \sigma(q) \in \Sigma$, and a positive integer l, we define $Len(l)(p)(q)$ as a set of lengths $|x|$ such that $p \xrightarrow{x} q$ and $|x| \leq l$. Let $m = |V|$. For a given M and l, we can compute the array *Len* efficiently in the following way. At first, we can obtain a new graph M' such that $L(M) = L(M')$ by deleting all the vertices p such that $\sigma(p) = \epsilon$ and rearranging the edges. This can be done in $O(m^3)$ time. Then, we can compute the array *Len* by modifying a fundamental algorithm for all pairs shortest paths problem and applying it to M'. This can be done in $O(m^3 l \log l)$ time. Therefore, the total time complexity for constructing the array *Len* is $O(m^3 l \log l)$.

3 Substructures and Boundary Contexts

3.1 Substructures of Multiple Loops

Let k be an integer with $k \geq 2$ and α_i $(i = 0, ..., k)$ be strings over Σ such that $|\alpha_0| = |\alpha_k| = 1$ and $|\alpha_i| \geq 2$ $(i = 1, ..., k-1)$. Then, by $(\overrightarrow{\alpha_0} \bullet \overrightarrow{\alpha_1 \bullet \cdots \bullet \alpha_{k-1}} \bullet \overrightarrow{\alpha_k})$, we denote a substructure of a multiple loop c consisting of: (1) k successive base pairs accessible from the closing base pair of c, which are base pairs between $\alpha_i[|\alpha_i|]$ and $\alpha_{i+1}[1]$ for $i = 0, ..., k-1$, and (2) sequences $\alpha_i[2, |\alpha_i|-1]$ of unpaired bases for $i = 1, ..., k-1$. This substructure is called a *multiple loop component*, or an *ML-component* for short. An ML-component is said to be *basic* if $k = 2$. (See Figure 1 (b).)

Let $a_1, a_2 \in \Sigma$ and α be a string over Σ such that $|\alpha| \geq 2$. An *ML-3'-end component*, written $(\overleftarrow{a_1} \bullet \overleftarrow{\alpha} \mid \overleftarrow{a_2})$, is defined as a substructure of a multiple loop c consisting of: (1) the closing base pair *bp* of c between $\alpha[1]$ and a_2 and a base pair between a_1 and $\alpha[|\alpha|]$ which is adjacent to *bp* and located at 3'-end direction from $\alpha[1]$, and (2) a sequence $\alpha[2, |\alpha|-1]$ of unpaired bases. An *ML-5'-end component*, written $(\overleftarrow{a_1} \mid \overleftarrow{\alpha} \bullet \overleftarrow{a_2})$, is defined as a substructure of a multiple loop c consisting of: (1) the closing base pair *bp* of c between $\alpha[|\alpha|]$ and a_1 and a base pair between a_2 and $\alpha[1]$ which is adjacent to *bp* and located at 5'-end direction from $\alpha[|\alpha|]$, and (2) a sequence $\alpha[2, |\alpha| - 1]$ of unpaired bases. (See Figure 1 (b).)

For convenience of the proofs, we will define the free energy of these substructures. For an ML-component $(\overrightarrow{\alpha_0} \bullet \overrightarrow{\alpha_1 \bullet \cdots \bullet \alpha_{k-1}} \bullet \overrightarrow{\alpha_k})$, we define:

$$E(\,(\overrightarrow{\alpha_0} \bullet \overrightarrow{\alpha_1 \bullet \cdots \bullet \alpha_{k-1}} \bullet \overrightarrow{\alpha_k})\,) =$$
$$\Sigma_{i=1}^{k-1}(\; m_1(\alpha_i[1], \alpha_{i-1}[|\alpha_{i-1}|], \alpha_i[2]) + m_2(\alpha_{i+1}[1], \alpha_i[|\alpha_i|], \alpha_i[|\alpha_i|-1]) +$$
$$M_u * (|\alpha_i| - 2) + M_b\;).$$

Recall that the third arguments of m_1 and m_2 should be unpaired bases. Note that in the case that there does not exist an unpaired base which is adjacent to a given base pair, ϵ is used as an argument of those functions.

Further note that the base pair contribution M_b of the last base pair between $\alpha_{k-1}[|\alpha_{k-1}|]$ and $\alpha_k[1]$ is not considered in this free energy definition.

For an ML-3'-end component $(\overleftarrow{a_1} \bullet \overleftarrow{\alpha} \mid \overleftarrow{a_2})$, we define:

$$E((\overleftarrow{a_1} \bullet \overleftarrow{\alpha} \mid \overleftarrow{a_2})) = m_1(\alpha[1], a_2, \alpha[2]) + m_2(a_1, \alpha[|\alpha|], \alpha[|\alpha|-1]) + M_u * (|\alpha| \ - \ 2) + M_b.$$

For an ML-5'-end component $(\overleftarrow{a_1} \mid \overleftarrow{\alpha} \bullet \overleftarrow{a_2})$, we define:

$$E((\overleftarrow{a_1} \mid \overleftarrow{\alpha} \bullet \overleftarrow{a_2})) = m_1(\alpha[1], a_2, \alpha[2]) + m_2(a_1, \alpha[|\alpha|], \alpha[|\alpha|-1]) + M_u * (|\alpha| \ - \ 2) + M_b.$$

Note that the base pair contribution M_b of the base pair containing the base a_1 is not considered also in these free energy definitions.

3.2 Contexts of Substructures

Let R be a regular language over Σ, and $M = (V, E, \sigma, I, F)$ be a trimmed graph accepting R. Let α (T) be a structured string such that $\alpha \in R$ and T is pseudo-knot free. Let us consider a base pair $(i_1, i_2) \in T$ of α (T). Let ρ be an accepting path of α. Then, (i_1, i_2) is said to have *context* (p_1, p_2) *in* ρ if for $l = 1, 2$, i_lth base of α is generated at the vertex p_l on ρ. For a base pair bp and an accepting path ρ, by $cf(bp, \rho)$ we denote the context of bp in ρ. In case that ρ is clear from the context, we omit the argument ρ and simply write $cf(bp)$. For context $v = (p_1, p_2)$, we often prefer to use the following representation:

$$v = (\uparrow p_1 \bullet p_2 \downarrow)$$

that is more graphically appealing to the reader. Note that for the context v, $(\sigma(p_1), \sigma(p_2)) \in \theta$ and $p_1 \rightarrow p_2$ should hold. By $Cfg(M)$, we denote the set of such well-defined contexts constructed from M. More formally, we define:

$$Cfg(M) = \{ (\uparrow p_1 \bullet p_2 \downarrow) \ \mid \ p_1, p_2 \in V, \ (\sigma(p_1), \sigma(p_2)) \in \theta, \ p_1 \rightarrow p_2 \}.$$

A 1-cycle c with a base pair bp in α (T) is said to have *context* v in ρ if $v = cf(bp, \rho)$. A 2-cycle c with base pairs bp_1 and bp_2 $(bp_1 > bp_2)$ in α (T) is said to have *context* (v_1, v_2) in ρ if $v_i = cf(bp_i, \rho)$ $(i = 1, 2)$. An ML-component in α (T) with base pairs bp_1, ..., bp_k $(bp_1 \prec \cdots \prec bp_k)$ is said to have *context* $(v_1, ..., v_k)$ in ρ if $v_i = cf(bp_i, \rho)$ holds for $i = 1, ..., k$. It is also said to have *boundary context* (v_1, v_k) in ρ. The *context* of an ML-3'-end component or an ML-5'-end component in ρ is defined as the pair (v_1, v_2) of contexts in ρ, where v_1 is a context of the closing base pair in the corresponding multiple loop and v_2 is a context of the other base pair in that substructure.

Definition 1. Let us consider two contexts $v_1 = (\uparrow p_1 \bullet q_1 \downarrow)$ and $v_2 = (\uparrow p_2 \bullet q_2 \downarrow)$ in $Cfg(M)$. Then, we define:

(1) $minH(v_1) = \mathbf{min}\,\{E(\overline{\uparrow x \downarrow}) \mid p_1 \xrightarrow{x} q_1,\ |x| \geq 2\}$,

(2) $minI(v_1, v_2) = \mathbf{min}\,\{E(|\uparrow\!x\!\!\!\!y\!\downarrow|) \mid p_1 \xrightarrow{x} p_2,\ q_2 \xrightarrow{y} q_1,\ |x| \geq 2,\ |y| \geq 2\}$,

(3) $minM(v_1, v_2) = \mathbf{min}\,\{E(\overrightarrow{(\sigma(p_1)} \bullet \overrightarrow{x} \bullet \overrightarrow{\sigma(q_2))}) \mid q_1 \xrightarrow{x} p_2,\ |x| \geq 2\}$,

(4) $minM_{3'}(v_1, v_2) = \mathbf{min}\,\{E(\overleftarrow{(\sigma(q_2)} \bullet \overleftarrow{x} \mid \overleftarrow{\sigma(q_1))}) \mid p_1 \xrightarrow{x} p_2,\ |x| \geq 2\}$,

(5) $minM_{5'}(v_1, v_2) = \mathbf{min}\,\{E(\overleftarrow{(\sigma(p_1)} \mid \overleftarrow{x} \bullet \overleftarrow{\sigma(p_2))}) \mid q_2 \xrightarrow{x} q_1,\ |x| \geq 2\}$. □

The existence of these minimum values is not clear at all. However, we can prove the existence of those minimum values in the following Theorem 1.

Recall the constants L_1, L_2, L_3 defined at Section 2.2 and the array Len introduced at the last paragraph of Section 2.3. Let $L = \mathbf{max}\,\{L_1, L_2, L_3\}$ and $m = |V|$. Furthermore, for $p, q \in V$, $a, b \in \Sigma$, and a positive integer l $(l \geq 3)$, we define:

$$\overline{Len}(l)(p)(q)(a)(b) = \bigcup_{\substack{(p,p'),(q',q) \in E \\ \sigma(p') = a,\ \sigma(q') = b}} \{x + 2 \mid x \in Len(l-2)(p')(q')\}.$$

This array $\overline{Len}(l)(\cdot)(\cdot)(\cdot)(\cdot)$ can be computed in $O(m^4 l)$ time from the array $Len(l-2)(\cdot)(\cdot)$.

Theorem 1. For contexts, $v_1 = (\uparrow p_1 \bullet q_1 \downarrow)$ and $v_2 = (\uparrow p_2 \bullet q_2 \downarrow)$ in $Cfg(M)$, the following equations hold:

(1) $minH(v_1) = \mathbf{min}\,\{E(\overline{\uparrow x \downarrow}) \mid p_1 \xrightarrow{x} q_1,\ 2 \leq |x| \leq L + m + 4\}$,

(2) $minI(v_1, v_2) = \mathbf{min}\,\{E(|\uparrow\!x\!\!\!\!y\!\downarrow|) \mid p_1 \xrightarrow{x} p_2,\ q_2 \xrightarrow{y} q_1,$
$2 \leq |x| \leq L + m^2 + m + 4,\ 2 \leq |y| \leq L + m^2 + m + 4\}$,

(3) $minM(v_1, v_2) = \mathbf{min}\,\{E(\overrightarrow{(\sigma(p_1)} \bullet \overrightarrow{x} \bullet \overrightarrow{\sigma(q_2))}) \mid q_1 \xrightarrow{x} p_2,\ 2 \leq |x| \leq m + 4\}$,

(4) $minM_{3'}(v_1, v_2) = \mathbf{min}\,\{E(\overleftarrow{(\sigma(q_2)} \bullet \overleftarrow{x} \mid \overleftarrow{\sigma(q_1))}) \mid p_1 \xrightarrow{x} p_2,$
$2 \leq |x| \leq m + 4\}$,

(5) $minM_{5'}(v_1, v_2) = \mathbf{min}\,\{E(\overleftarrow{(\sigma(p_1)} \mid \overleftarrow{x} \bullet \overleftarrow{\sigma(p_2))}) \mid q_2 \xrightarrow{x} q_1,$
$2 \leq |x| \leq m + 4\}$,

Furthermore, for given contexts, the above minimum values can be computed in $O(m^4)$ time using the information about the array $\overline{Len}$. □

4 Algorithm for Testing Structure Freeness

Let M be a reduced graph representing a given regular set R and consider context set $Cfg(M)$. Let us consider a context $v = (\uparrow p \bullet q \downarrow) \in Cfg(M)$ and a structured string α (T) such that $p \xrightarrow{\alpha} q$ and $(1, |\alpha|) \in T$. Then, this structure

$\alpha\ (T)$ is called a *closed structure with boundary context* v. The base pair $(1, |\alpha|)$ is called an *end closing* base pair of the closed structure $\alpha(T)$.

For $v = (p, q) \in Cfg(M)$, by $\delta(v)$ we denote the minimum free energy among all closed structures with boundary context v. More formally, we define:

$$\delta(v) = \mathbf{min}\,\{\, E(\alpha\ (T)) \mid p \overset{\alpha}{\to} q,\ (1, |\alpha|) \in T\,\}.$$

In case that such minimum free energy does not exists, we define $\delta(v) = -\infty$. Since we assume that the free energy of any free end substructure is always 0, it is enough for us to compute $\delta(v)$ for all $v \in Cfg(M)$ and check whether or not for every $v \in Cfg(M)$, $\delta(v) \geq 0$ holds.

4.1 Minimum Free Energy of ML-Components Under Energy Assignment

An *energy assignment* C is a mapping from $Cfg(M)$ to the set of real values. Let us consider an ML-component $c = (\overrightarrow{\alpha_0} \bullet \overrightarrow{\alpha_1 \bullet \cdots \bullet \alpha_{k-1}} \bullet \overrightarrow{\alpha_k})$ in a structured string $\alpha\ (T)$ and let bp_i $(i = 1, ..., k)$ be the ith base pair of c from the 5'-end (i.e. a base pair between $\alpha_{i-1}[|\alpha_{i-1}|]$ and $\alpha_i[1]$). Let ρ be an accepting path of α. Let v_i be a context of bp_i in ρ $(i = 1, ..., k)$, and let $cnf = (v_1, ..., v_k)$. The *free energy* $\tilde{E}(c, cnf, C)$ *of* c *with respect to* cnf *under energy assignment* C is defined as $\tilde{E}(c, cnf, C) = E(c) + \Sigma_{i=1}^{k-1} C(v_i)$. Note that $C(v_k)$ is not contained in this definition. In case that $cnf = (v_1, ..., v_k)$ is clear from the context, we will often omit the second argument cnf in $\tilde{E}(c, cnf, C)$ and simply write $\tilde{E}(c, C)$.

An ML-component c with context $cnf = (v_1, ..., v_k)$ is said to be *E-minimal with respect to* cnf *under energy assignment* C if for any ML-component c' with context cnf, $\tilde{E}(c, C) \leq \tilde{E}(c', C)$ holds.

Now, we have interests to compute for given contexts v_1 and v_2, the minimum free energy $\Delta(v_1, v_2, C)$ under energy assignment C among all ML-components with boundary context (v_1, v_2).

This problem can easily be reduced to the problem of all pairs shortest paths of a graph defined bellow.

For given context set $Cfg(M)$ and energy assignment C, we construct a weighted directed graph $G(M, C) = (V_*, E_*, w)$ defined by: $V_* = Cfg(M)$, $E = V_* \times V_*$, and $w((v_1, v_2)) = minM(v_1, v_2) + C(v_1)$ for all $v_1, v_2 \in V_*$. Then, we can show the following theorem.

Theorem 2. *The weight of a path* $v_1, ..., v_k$ *in* $G(M, C)$ *is equal to the free energy of an E-minimal ML-component with respect to* $(v_1, ..., v_k)$ *under* C. □

If there exists a negative weight cycle, and there exists no shortest path from v_1 to v_2, then the weight of the shortest path from v_1 to v_2 is defined as $-\infty$. Then, we have the following corollary.

Corollary 1. *The weight of the shortest path from* v_1 *to* v_2 *in* $G(M, C)$ *gives the value* $\Delta(v_1, v_2, C)$.

Proof
Note that the minimum value $\Delta(v_1, v_2, C)$ is given by an E-minimal ML-component. □

```
$\mathbf{SFT}(M)$
begin
    compute $Len$ up to $l = L + m^2 + m + 4$;
    compute $\overline{Len}$ for $l = m + 4,\ L + m + 4,\ L + m^2 + m + 4$;
    compute $minH$, $minI$, $minM$, $minM_{3'}$, $minM_{5'}$ for all contexts;
    for $i = 1$ to $n$ do  $C_1(v) := minH(v)$; end
    for $i = 2$ to $n + 1$ do  $C_i :=\mathbf{Update}(C_{i-1})$;  end
    if there exists $v \in Cfg(M)$ such that $C_n(v) < 0$ or $C_{n+1}(v) < C_n(v)$  then
        return 'NO';
    else return 'YES';
end

$\mathbf{Update}(C)$
begin
    $D :=$ $\mathbf{APSP}(G(M, C))$;
    if($D$ =='NO') then output 'NO' and halt;
    for $v \in Cfg(M)$ do
        $X := \mathbf{min}\{\ C(v') + minI(v, v')\ \mid\ v' \in Cfg(M)\ \}$;
        $Y := \mathbf{min}\{\ minM_{3'}(v, v') + D(v', v'') + C(v'') + minM_{5'}(v, v'') + C_M \quad |$
                    $v', v'' \in Cfg(M)\ \}$;
        $C'(v) := \mathbf{min}\{\ C(v),\ X,\ Y\ \}$;
    end
    return $C'$;
end
```

Fig. 2. Proposed Algorithm **SFT**

For instance, we can use the Johnson's algorithm ([Joh77]) for computing all pairs shortest paths. Johnson's algorithm, denoted **APSP** in the sequel, allows the use of negative weight edge and can detect negative weight cycle. It returns 'NO' if there exists a negative weight cycle, otherwise returns a distance matrix between the vertices. The time complexity of his algorithm is $O(\,|V_*|^2 \log |V_*| + |V_*||E_*|\,)$.

4.2 Algorithm SFT

We are ready to give an algorithm **SFT** for testing the structure freeness of a given regular set R. Let $M = (V, E, \sigma, I, F)$ be a graph representing R, $n = |Cfg(M)|$, and $m = |V|$. Then, the proposed algorithm is shown in Figure 2.

Theorem 3. Algorithm **SFT** is correct. The time complexity of **SFT** is $O(m^8)$, where m is the number of vertices of M. □

References

Adl94. L. Adleman, Molecular Computation of Solutions to Combinatorial Problems. *Science* **266**, pp.1021-1024, 1994.

ADS02. M. Andronescu, D. Dees, L. Slaybaugh, Y. Zhao, A. Condon, B. Cohen, S. Skiena, Algorithms for testing that DNA word designs avoid unwanted secondary structure, In *Proc. of 8th International Meeting on DNA Based Computers*, pp.92-104, 2002.

BC01. A. Brenneman, A. E. Condon, Strand Design for Bio-Molecular Computation, *Theoretical Computer Science*, **287**, pp.39-58, 2002.

Con03. A. E. Condon, Problems on RNA Secondary Structure Prediction and Design, *Proc. of ICALP'2003*, Lecture Notes in Computer Science, Vol.2719, pp.22-32, 2003.

HFS94. I.L. Hofacker, W. Fontana, P.F. Stadler, L.S. Bonhoeffer, M. Tacker, P. Schuster, Fast Folding and Comparison of RNA Secondary Structures (The Vienna RNA Package), *Monatshefte für Chemie*, **125**, pp.167-188, 1994.

Joh77. D. S. Johnson, Efficient Algorithms for Shortest Paths in Sparse Networks, *Journal of the ACM*, Vol.24, pp.1-13, 1977.

KYS04. S. Kobayashi, T. Yokomori, and Y. Sakakibara, An Algorithm for Testing Structure Freeness of Biomolecular Sequences, in *Aspects of Molecular Computing — Essays dedicated to Tom Head on the occasion of his 70th birthday*, Springer-Verlag, LNCS **2950**, pp.266-277, 2004.

SKM83. D. Sankoff, J.B. Kruskal, S. Mainville, R.J. Cedergren, Fast Algorithms to Determine RNA Secondary Structures Containing Multiple Loops, in *Time Warps, String Edits, and Macromolecules : The Theory and Practice of Sequence Comparison*, D. Sankoff and J. Kruskal, Editors, Chapter 3, pp.93-120, 1983.

ZS81. M. Zuker, P. Steigler, Optimal Computer Folding of Large RNA Sequences using Thermodynamics and Auxiliary Information, *Nucleic Acids Research*, **9**, pp.133-148, 1981.

Minimum Basin Algorithm: An Effective Analysis Technique for DNA Energy Landscapes

Mitsuhiro Kubota and Masami Hagiya

Japan Science and Technology Agency (JST-CREST),
Department of Computer Science,
Graduate School of Information Science and Technology,
University of Tokyo,
7-3-1 Hongo, Bunkyo-ku, Tokyo 113-0033, Japan
{kubota, hagiya}@is.s.u-tokyo.ac.jp

Abstract. To design DNA nano-machines or analyze DNA molecular reactions, it is important to be able to predict the energy landscape of molecular structures and the energy barrier of a transition between structures on the landscape. Unfortunately, this is difficult for DNA molecules longer than 100 bases. In this paper, we propose an effective new technique for analyzing a structural transition over a DNA energy landscape. Imagine a very undulating landscape. Suddenly, water starts to gush out from one site and keeps flowing. How will the water surface expand over the landscape? Using a variant of Dijkstra's and Jarník-Prim's algorithms, we generate the shape of the basin from its formation process. The resulting basin contains the true energy barrier. Furthermore, a comparison between the basin feature and the corresponding actual chemical reaction shows that the basin can be used as a criterion to explain the reaction.

1 Introduction

In the field of molecular programming [11], which aims to establish general principles for building molecular systems, sequence design and the analysis of molecular reactions are among the most important research themes. As one approach to these problems, research using the thermodynamics of secondary structures is flourishing [1, 2, 3, 5, 12, 17].

As a method of computing the free energy of a nucleic acid secondary structure, the model based on nearest neighbor thermodynamics [21] is widely used. This model assigns a free energy parameter obtained experimentally to each loop surrounded by the backbone and the hydrogen bonds between the bases of the nucleic acid. The energy of the entire secondary structure is approximated by the sum of such free energies. For DNA, the parameters are provided by SantaLucia et al. [19] among others. According to the model, when a base sequence is given, the minimum free energy structure (mfe structure), which is the most stable structure in solution, can be predicted. The problem of obtaining the mfe structure is called the folding problem and is solved effectively using dynamic programming [26]. Conversely, when a secondary structure is given, the problem of finding a sequence whose mfe structure is the given one is called the inverse folding problem. This can be formulated as an optimization problem corresponding to the folding [13].

Furthermore, in some cases, we want to consider not only the minimization of the free energy but also the maximization of the probability that the given structure is assumed. For this purpose, it is necessary to obtain the partition function, which can also be calculated using dynamic programming [14]. Moreover, a method of enumerating suboptimal structures in addition to the mfe has been studied [24].

Recently, a DNA molecular machine based on the conformational changes caused by fuel DNA via branch migration was reported [25]. A model based on nearest neighbor thermodynamics could also be used to design such a molecular machine. By using and extending the model, Uejima et al. designed their "hairpin-based multi-state machine", which is based on the sequential opening of DNA hairpin structures, with the intention of developing more general multi-state machines [22]. They proposed an application of their machine to molecular memory, called "conformational addressing", and verified its reactions [16]. In the design, they considered not only static equilibrium energies but also structural transition paths, because some operations of their machine depend on free energy traps (local minima) and they need to evaluate the (in)efficiency of structure transitions out of a trap.

In order to evaluate a transition between two structures, it is necessary to define a path connecting the two structures and to calculate the height of the energy barrier on the path. To predict the transition path, Uejima et al. employed the heuristics by Morgan and Higgs [15], which have a superior computational efficiency compared with other methods. This heuristics restrict the transition paths to those called "direct paths", which consist of steps that remove the base pairs in the start structure that do not exist in the target structures and steps that add base pairs in the target structure that do not exit in the start structure.

Previously, we designed and verified our hairpin-based machine, called the "branching machine", whose state can branch according to the input. In the design, we also followed Uejima's footsteps and used Morgan-Higgs' heuristics [18]. As for direct paths, a breadth-first search algorithm [7] and an improvement of Morgan-Higgs' heuristics [23] have been proposed.

However, the structures on a direct path only contain base pairs that exist in either the start or target structure. This condition obviously makes the analysis based on transition paths restrictive. In order to analyze the energy landscape, especially to seek the true energy barrier, it is also necessary to consider base pairs that do not exist in the start or target structures.

For this purpose, analyses using the Markov process and a kind of Monte Carlo methods were studied by Flamm et al. [8], and the flooding algorithm was proposed by Stadler and Flamm [20]. The flooding algorithm generates not only the true energy barrier but also the barrier tree that is a fine representation of the energy landscape [6]. However, this algorithm needs to enumerate all the structures that the given sequence can assume and to calculate their energies in advance. Because the number of all possible structures increases exponentially with the sequence length, it is difficult to enumerate and calculate them all when the length exceeds 100 bases [4]. Even if this were possible, the energies of almost all of the structures are quite high [4] and useless.

In this paper, we notice that the analysis of the energy geographical features of DNA is a special case of the minimax path problem and propose a technique for the analysis based on the fact that the minimax path is a path on the minimum spanning tree. The technique is a variant of Dijkstra's and Jarník-Prim's algorithms, which are used to obtain the shortest path and minimum spanning tree, respectively. An experimental result of our "branching machine" shows that this technique is reasonable for explaining an actual reaction.

2 Advanced Analysis Technique for Energy Landscapes

Consider a DNA molecule that undergoes a change from one conformation S_s *:start* to another S_t*:target*. In this study, a structural transition is defined as a sequence involving the "formation (addition) of one base pair" or "dissociation (removal) of one base pair." Then, there must be a structure S_b*:barrier* that gives the energy barrier on the path.

If the molecule can attain the conformation S_t, then the probability that the molecule carries out a conformational change via the path with the lowest energy barrier is high. To design a DNA machine using a conformational change, it is reasonable to use DNA sequences with the high or low energy barrier depending on the specifications.

Because one can search for a structure adjacent to a given structure by adding or removing one base pair, we can consider the graph $G = (V, E)$ in which a vertex expresses a structure and an edge expresses an adjacency relation between each pair of vertices. Then, the algorithm begins with the set which consists only of the initial structure S_s and extends the set using the adjacency relation. It ends when the set includes the target structure S_t.

Minimum Basin Algorithm

```
1) Given start vertex $S_s$ and target vertex $S_t$.
2) $\mathcal{B}$, $\mathcal{N}$:= $\emptyset$.     $\mathcal{B}$: a set of reachable, low energy vertices
                   $\mathcal{N}$: a set of vertices neighboring a vertex in $\mathcal{B}$
3) Add $S_s$ to $\mathcal{B}$.
4) Add $u \in N(v) \setminus (\mathcal{B} \cup \mathcal{N})$ to $\mathcal{N}$,
   where $v$ is added to $\mathcal{B}$ in the previous step.
   ($x \in N(y)$ iff $\exists e(x, y) \in E$.)
5) Move $v \in \mathcal{N}$ with the minimum energy to $\mathcal{B}$ from $\mathcal{N}$.
   (If there are multiple minimum energy vertices, then Move all of them.)
6) Go to Step 4. If $v = S_t$ in Step 5, then End.
```

Subset $\mathcal{B} \subseteq V$ in this algorithm contains structures in the *minimum basin* that connect the starting S_s and terminal S_t points. In other words, if we assume that the starting point S_s is a *source* and the terminal point S_t is a *switch to stop the water*, the set $\mathcal{B}$ consists of the structures that appear in the basin of water over the energy landscape. Because $\mathcal{B}$ covers all of the low energy structures that can be reached with a conformational change, it is suitable for comparison with the result of an experiment. Moreover, the structure with the maximum energy in $\mathcal{B}$ serves as the true energy barrier.

This technique can be used for any landscape in which each vertex can be defined using any weight, like energy, and the adjacency relation between each pair of vertices can be defined.

3 Relation to the Minimax Path Problem

The minimax path is the path whose maximum distance between two adjacent vertices is the minimum. For example, one needs this path to cross a vast desert via oases, choosing the path whose maximum distance between two oases connected directly on the path is the minimum. This situation resembles that of a conformational change via the path with the lowest energy barrier, except that while weights are given to edges in typical graphs, i.e., $E \rightarrow \mathbb{R}$, energies are assigned to vertices on the energy landscape, i.e., $V \rightarrow \mathbb{R}$.

Because the minimax path is a path on the minimum spanning tree, the algorithm used to obtain the path with the lowest energy barrier must be related to that of the minimum spanning tree. In fact, our minimum basin algorithm is a variant of the Dijkstra-Jarník-Prim (DJP) algorithm. In the DJP, the temporary shortest distance or temporary minimum edge is held at each vertex and this value is updated in these algorithms. Energy is assigned to each vertex on the energy landscape, and thus it is not necessary to update the value in the minimum basin algorithm. The flooding algorithm is also a variant of Kruskal's algorithm. According to the above-mentioned feature, it is not necessary to check a cycle in the flooding algorithm. Therefore, the analysis of the energy landscape is easier than that of the minimax path problem.

In the minimum spanning tree problem, the obtained result can be represented graphically by the taxonomic tree called the *dendrogram.* The barrier tree for the energy landscape is a variant of the dendrogram.

The minimum basin algorithm can obtain a kind of minimax path that has the lowest energy barrier using trace back. The path obtained is the reasonable path that is most stable energetically overall.

4 Efficient Data Structures for Implementation

When we use the minimum basin algorithm, the main procedure consisting of Steps 5 and 6 in Sect. 2 is that shown in Table 1. In order to implement this

Table 1. Main Repeat Procedure

1) Neighboring Structures
Specify a pair that can be added or removed,
in order to find one neighbor of the current structure.
2) Overlap Check
Confirm that the structure has not appeared before.
3) Calculate Energy
Calculate the energy of the new structure.
4) Compare Energy with Threshold
If the new energy is lower than the threshold,
then Add it to the priority queue.
5) Repeat
If neighboring structures remain, then Go to 1.
6) Select Minimum
Select a structure as the current structure for the next iteration.

Table 2. Strategies for Implementation

Data/Operation etc.	Data Structure etc.
Secondary Structure	Ordered Rooted Tree (Ring List Tree) Base Pair List
Check Overlap	Base Pair Tree
Calculate Energy	3-Loop Energy Difference
Loop Energy of Virtual Base	External Loop
Energy Threshold	Morgan-Higgs' Heuristics
Select Minimum (Neighboring Set $\mathcal{N}$)	Priority Queue (Incremental Binary Heap)
Multiple Minima	Breadth First
Minimum Basin $\mathcal{B}$	Final Basin List
Yield Path	Minimax Path

procedure, we used various strategies (Table 2). We present the details of the individual strategies in the next subsections.

Ring List Tree and Base Pair List. In order to choose a base pair to add or remove effectively (Table 1–Step 1), it is necessary to use good data structures representing secondary structures. We used both the *ring list tree* (Fig. 1–C) and the *base pair list* (Fig. 2–B).

In the case of addition, it is necessary to make a pair using bases that belong to the same loop, so that the resulting structure does not become a pseudoknot. A secondary structure can be converted into a tree representation called an *ordered rooted tree* [10, 13]. The *ring list tree* we use is a kind of ordered rooted tree, which Flamm et al. implemented in their stochastic simulation based on

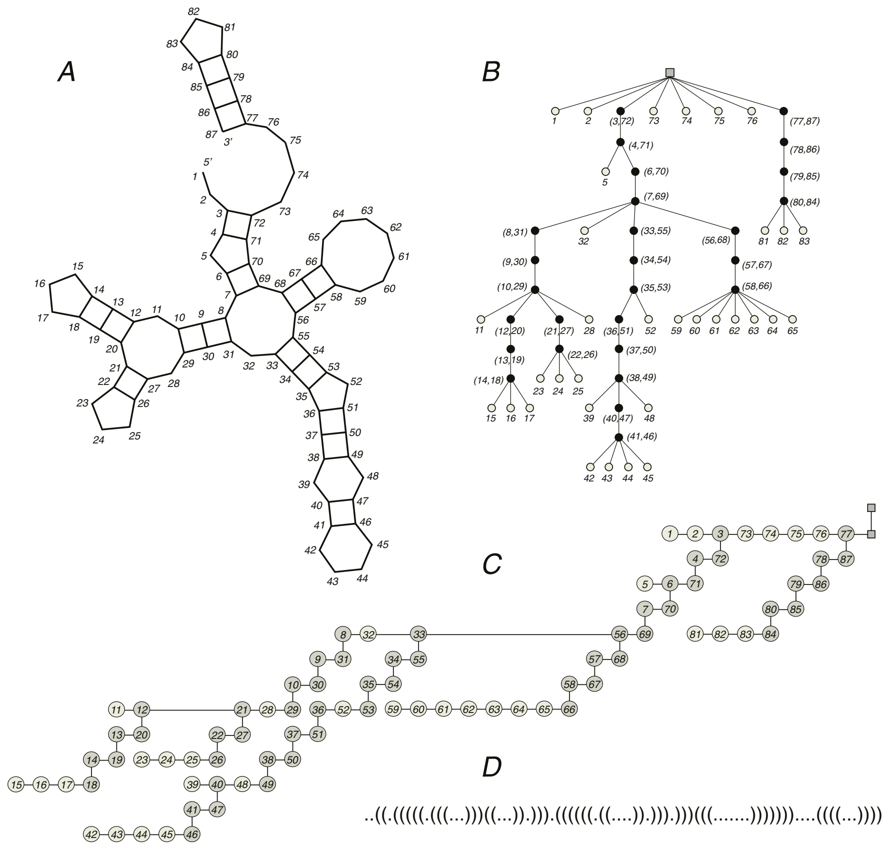


Fig. 1. Various Representations of a Secondary Structure. A: Planner Graph, B: Ordered Rooted Tree, C: Ring List Tree, D: Dot-Parenthesis Notation

A	B
((..((....))....))	(1,18)-(2,17)-(5,12)-(6,11).
((.((......))...))	(1,18)-(2,17)-(4,13)-(5,12).
((.(((....)))...))	(1,18)-(2,17)-(4,13)-(5,12)-(6,11).
((((((....))).).))	(1,18)-(2,17)-(3,15)-(4,13)-(5,12)-(6,11).

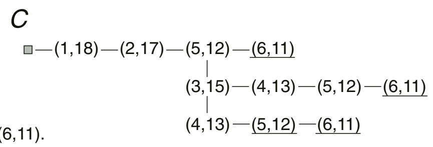


Fig. 2. Base Pair Tree and Multiple Secondary Structures. A: Dot-Parenthesis Notation, B: Base Pair List, C: Base Pair Tree

the Monte Carlo method [9, 8]. This tree uses horizontal lines to represent bases that belong to the same loop. Therefore, using the ring list tree, we can find neighboring structures effectively in the case of addition.

On the other hand, in the case of removal, we must scan the entire tree. To overcome this problem, it is effective to use a list representation. Each node in the list is connected according to the order of indices of base pairs. We call this list the *base pair list*. Removal can be realized by deleting a node from the list.

Base Pair Tree. In Step 2 in Table 1, we must check whether each calculated neighboring structure has already appeared. A mechanism that keeps the structures that have already appeared is needed for this overlap check.

We propose the *base pair tree* (Fig. 2–C), which represents multiple base pair lists. Each node of this tree also represents a base pair. By choosing a node, we can decide the unique structure consisting of the base pairs on the path traced back to the root of the tree.

Each node has a Boolean value: 0 or 1. The tree is traversed using the base pair list of a given neighboring structure. If the value of the final node in the traversal is 0, then the neighboring structure has not yet appeared. If the value is 1, the structure has already appeared. In any case, the value of the final node is changed to 1.

3-Loop Energy Difference. In Step 3 in Table 1, we must calculate the energy of the new structure. We use the fact that an energy parameter is assigned to each loop in the nearest neighbor model. For structures obtained using the adjacency relation, the energy difference between two neighboring structures arises from the loops before and after a transition. The number of different loops is *always three* (Fig. 3). Therefore, it is sufficient to calculate this difference. The ring list tree plays an important role in this calculation.

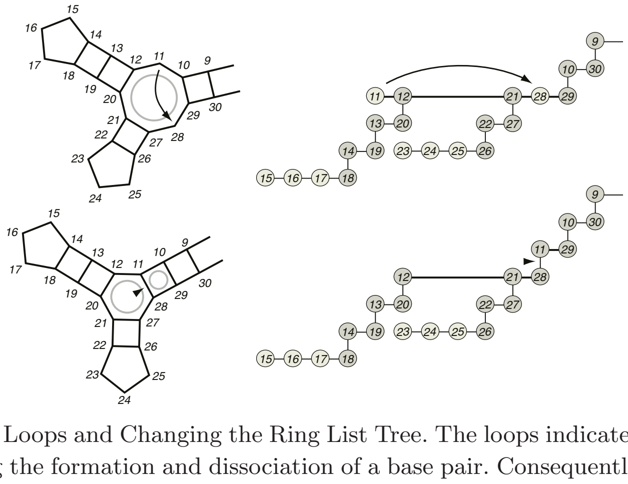


Fig. 3. Three Loops and Changing the Ring List Tree. The loops indicated by the cycle change during the formation and dissociation of a base pair. Consequently, the ring list tree is manipulated

Loop Energy for Virtual Bases. In order to calculate the energy of a secondary structure formed by two or more sequences, these sequences must be connected using *virtual bases* that do not form base pairs with other bases. Uejima et al. calculated a loop energy parameter including virtual bases using the equilibrium analysis of reactions consisting of two molecules [22], because it was necessary to compare the energy difference between separated and hybridized molecules. Ackermann et al. used an approximation with sixteen virtual bases [1].

We follow Uejima's approach for our analysis, while considering the energy of the dangling ends. For a single sequence, the loop containing the 5'- and 3'-ends belongs to the external loop. Therefore, it is reasonable to consider that a loop containing virtual bases is also an external loop, otherwise there would be an energy difference between the two ways of connecting two DNA sequences using virtual bases, i.e., `5'-A-spacer-B-3'` and `5'-B-spacer-A-3'`, and each way would produce a different mfe structure.

Morgan-Higgs' Heuristics and Its Modification. Morgan and Higgs proposed that their heuristics can be used to give a threshold for the true energy barrier, i.e., $Barrier_{true} \leq Barrier_{MH}$ [15]. It is also an effective way to reduce the search space in our algorithm (Table 1–Step 4). Structures whose energies exceed the barrier obtained using the heuristics need not be considered for selection and can be discarded.

In using Morgan and Higgs' heuristics, we made the following small modification. At the end of the heuristics, we remove the base pairs remaining from the initial structure. This is simply because in their original heuristics, the base pairs of the initial structure that are not incompatible with the final one are never chosen. Because the Morgan-Higgs' heuristics keeps as many base pairs in the initial structure as possible, it is reasonable to remove the remaining base pairs at the end.

Multiple Minimum Energy Structures. The minimum energy structure of $\mathcal{N}$ is not unique in general (Table 1–Step 6). If we only need to know the energy barrier involved in a conformational change, it is sufficient to select only one of the multiple minimum energy structures for the next iteration. This results in the depth-first strategy, which is more effective for obtaining the barrier.

To analyze the minimum basin $\mathcal{B}$, however, we should adopt the breadth-first strategy. Using the depth-first strategy, if the algorithm stops when the basin reaches the target structure S_t, then the basin obtained is smaller than the true minimum basin and this depends on the selection of the minimum energy structure. By choosing all of the minimum energy structures, we can simulate water gushing over an undulating landscape.

5 Analysis Using the Minimum Basin

Our branching machine is a single DNA molecule consisting of two hairpin structures connected by a single-stranded section. The input DNA can invade a hairpin's stem via branch migration, opening the hairpin. The branching state is represented by which of hairpins is opened. In Fig. 4, conformation (b) corresponds to one of the branching states, while the change to (c) is undesirable. Therefore, our criteria were to maximize the barrier height B on the path from (b) to (c) and to minimize the energy valley depth V for both the left and right

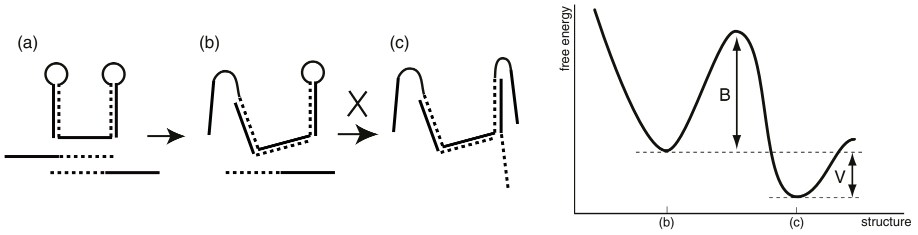


Fig. 4. Conformational Change of Our Branching Machine. The graph on the right side shows the free energy following the conformational change from (b) to (c)

Table 3. Sequences of the Branching Machine and Input Oligomer

Branching Machine (Hairpin)	:	`5'-TATAAAACCCTATCTATGCG-ACACATA-CGCATAGATAGGGTTTTATA-`
		`-CCGCACGAGACCCCACCCTC-`
		`-CGCGCAAGAACCATTTGTTG-CAGCGCA-CAACAAATGGTTCTTGCGCG-3'`

Open 5	:	`5'-GAGGGTGGGGTCTCGTGCGG-TATAAAACCCTATCTATGCG-3'`
Open 3	:	`5'-CAACAAATGGTTCTTGCGCG-GAGGGTGGGGTCTCGTGCGG-3'`

sides. Table 3 shows the best sequences in our design using Morgan-Higgs' heuristics. Open 5 opens the hairpin's 5'-side and Open 3 opens the hairpin's 3'-side. The hairpin is 114 bases long and the input DNA is 40 bases.

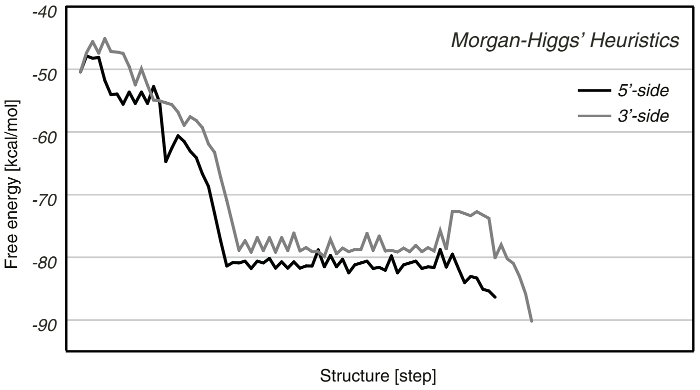
An experiment showed that the 5'-side input DNA was much better at hybridizing with the branching machine than the 3'-side [18]. We postulated that the mfe structure of the input DNA influences the reaction.

Fig. 5 shows the free energies along the structural transition paths obtained using the Morgan-Higgs' and our minimum basin algorithms.

On the Morgan-Higgs' paths, the 3'-side energy barrier is higher than that on the 5'-side. During the structural transition after the nucleation for the hybridization, both sides must assume a structure that is more unstable than structure S_s. This results in a big barrier. In this way, using Morgan-Higgs' heuristics, we can discuss the energy barrier on the direct paths. However, it does not tell us about the true energy barrier or the structures in the middle of the transition.

By contrast, on the minimax paths obtained using the minimum basin algorithm, we can find the true energy barrier, which does not usually appear on direct paths. It turned out that it was unnecessary to climb over a big barrier on both sides. However, a structure considered to be a kind of barrier still exists on both sides. The energy of the structure on the 3'-side is higher than that on the 5'-side. Furthermore, using the minimum basin algorithm, we can analyze the minimum basin $\mathcal{B}$ and obtain the structures in the middle of the transition.

Table 4 shows some features of the minimum basin $\mathcal{B}$ and minimax path. In the first column is the number of structures comprising the minimum basin. The next column shows the *barrier height* B_c from S_s to the structure considered to be a kind of barrier, the *number* of structures that are reached before the barrier structure, and the *ratio* of these structures with respect to the minimum basin.



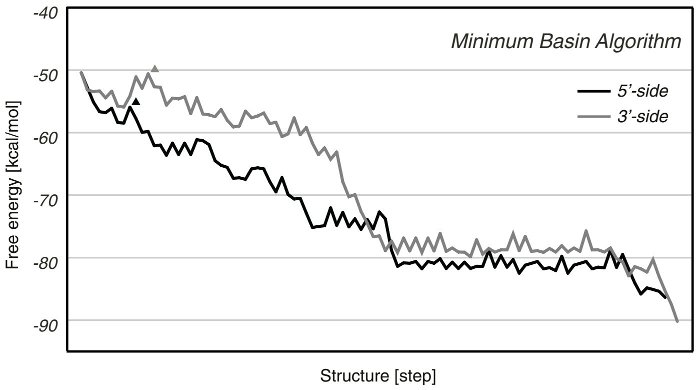


Fig. 5. Free Energies along the Structural Transition Path. These are the paths from the mfe structure when the hairpin does not hybridize with any input DNA to the mfe structure when the hairpin hybridizes with one of the input DNA. The vertical axis is the free energy of the structure given on the horizontal axis. The "addition" or "removal" of a base pair is one step involved in changing the secondary structure along the horizontal axis. In the lower graph, the lower left corner of the triangle shows the structure that is thought to be a kind of barrier

Table 4. Features of the Minimum Basin and Path. (B_c [kcal/mol])

	Basin	Barrier (Maximum Convex)			Length	
	$\|\mathcal{B}\|$	$< B_c$	$\leq B_c$	B_c	Step	Dist.
5'-side	3416	2311 (67.7%)	2338 (68.5%)	−5.50	96	68
3'-side	65887	61801 (93.8%)	62548 (94.9%)	−0.17	98	74

The last column presents the lengths of the minimax path (Step) and the direct path (Dist.).

These results can explain the molecular reactions that make it comparatively more difficult for the 3'-side to occur than the 5'-side. Among several criteria in Table 4, the number of structures can be evaluated while calculating the minimum basin. We can set a threshold on the number of structures used to judge a given sequence; we can stop the algorithm when the number reaches the threshold.

For the 5'-side, it took 8–9 seconds (CPU time) and about 200 MB of memory to calculate the minimum basin. This performance was measured on a Pentium 3, 500 MHz, using Windows 2000 with 320 MB of memory. The sequence used for this calculation was 158 bases long (input + virtual bases + hairpin = 40 + 4 + 114). Although it is difficult to analyze the energy landscape of such a long sequence after enumerating all the structures (see [4] Table 1), it is reasonable to analyze it using the minimum basin algorithm. Using our original implementation, which did not use the strategies mentioned in Sect. 4, we spent over a day and about 10 MB of memory on the same problem. Therefore, those strategies are quite effective for a long sequence. Incidentally, even using the original implementation, we were able to analyze a sequence with a length of 20 bases in a instant.

6 Conclusion

We proposed the minimum basin algorithm and showed that the analysis of the minimum basin $\mathcal{B}$ obtained using this algorithm gives a reasonable criterion for explaining actual reactions. Robust molecular machines and systems should be realizable using sequence designs based on this technique. Furthermore, detailed analyses could be performed using the flooding algorithm for the minimum basin $\mathcal{B}$.

Despite the efforts of many researchers, the technique still involves several uncertainties, e.g., the accuracy of the parameters and approximations used by the algorithm. While such predictions are statistically accurate, they may not be as accurate in individual cases [8]. Therefore, in order to use this technique for sequence design, it is important to verify individual sequence reactions experimentally. In the future, it will be necessary to draw and collect the correspondence maps between reaction experiments and analytical predictions in order to incorporate the knowledge obtained from the map in sequence design.

In addition to DNA, the minimum basin algorithm should also be effective for any energy or fitness landscape. Presently, we are trying to apply this technique to analyze actual RNA reactions.

Acknowledgments

This work was supported by JST CREST and the Ministry of Education, Culture, Sports, Science, and Technology of Japan under Grants-in-Aid for Scientific Research on Priority Areas (B), 14085101 and 14085202, 2004.

References

1. Ackermann, J. et al.: Word design for molecular information processing. Zeitschrift für Naturforschung **58a** (2003) 157–161
2. Andronescu, M. et al.: RNAsoft: a suite of RNA secondary structure prediction and design software tools. Nucl. Acids. Res. **31** (2003) 3416–3422
3. Andronescu, M. et al.: Algorithms for testing that sets of DNA words concatenate without secondary structure. DNA8, Springer LNCS **2568** (2003) 182–195
4. Cupal, J. et al.: Density of States, Metastable States, and Saddle Points Exploring the Energy Landscape of an RNA Molecule. Proc. of 5th Int. Conference on Intelligent Systems for Molecular Biology (ISMB-97) (1997) 88–91
5. Deaton, R. et al.: A Software Tool for Generating Non-crosshybridizing Libraries of DNA Oligonucleotides. DNA8, Springer LNCS **2568** (2003) 252–261
6. Flamm, C. et al.: Barrier Trees of Degenerate Landscapes. Z. Phys. Chem. **216** (2002) 155–173
7. Flamm, C. et al.: Design of multistable RNA molecules. RNA **7** (2001) 254–265
8. Flamm, C. et al.: RNA folding at elementary step resolution. RNA **6** (2000) 325–338
9. Flamm, C.: Kinetic Folding of RNA. PhD Thesis, University of Vienna, Austria (1998)
10. Fontana, W. et al.: Statistics of RNA secondary structures. Biopolymers **33** (1993) 1389–1404
11. Hagiya, M.: Towards Molecular Programming. Modeling in Molecular Biology, Springer Natural Computing Series (2003) (in press)
12. Heitsch, C.E. et al.: From RNA Secondary Structure to Coding Theory: A Combinatorial Approach. DNA8, Springer LNCS **2568** (2003) 215–228
13. Hofacker, I.L. et al.: Fast Folding and Comparison of RNA Secondary Structures. Monatshefte für Chemie **125** (1994) 167–188
14. McCaskill, J.S.: The Equilibrium Partition Function and Base Pair Binding Probabilities for RNA Secondary Structure. Biopolymers **29** (1990) 1105–1119
15. Morgan, S.R. et al.: Barrier heights between ground states in a model of RNA secondary structure. J. Phys. A: Math. Gen. **31** (1998) 3153–3170
16. Kameda, A. et al.: Conformational addressing using the hairpin structure of single-strand DNA. DNA9, Springer LNCS **2943** (2004) 219–224
17. Kobayashi, S. et al.: An Algorithm for Testing Structure Freeness of Biomolecular Sequences. Aspects of Molecular Computing, Springer LNCS **2950** (2004) 266–277
18. Kubota, M. et al.: Branching DNA Machines Based on Transitions of Hairpin Structures. Proc. of the 2003 Congress on Evolutionary Computation (CEC2003) **4** (2003) 2542–2548
19. SantaLucia, J., Jr.: A unified view of polymer, dumbbell, and oligonucleotide DNA nearest-neighbor thermodynamics. Proc. Natl. Acad. Sci. USA **95** (1998) 1460–1465
20. Stadler, P.F. et al.: Barrier Trees on Poset-Valued Landscapes. J. Gen. Prog. Evol. Machines **4** (2003) 7–20
21. Tinoco, I., Jr. et al.: Estimation of secondary structure in ribonucleic acids. Nature **230** (1971) 362–367
22. Uejima, H. et al.: Secondary Structure Design of Multi-state DNA Machine Based on Sequential Structure Transitions. DNA9, Springer LNCS **2943** (2004) 74–85
23. Uejima, H. et al.: Analyzing Secondary Structure Transition Paths of DNA/RNA molecules. DNA9, Springer LNCS **2943** (2004) 86–90

24. Wuchty, S. et al.: Complete Suboptimal Folding of RNA and the Stability of Secondary Structures. Biopolymers **49** (1999) 145–165
25. Yurke, B. et al.: A DNA-fuelled molecular machine made of DNA. Nature **406** (2000) 605–608
26. Zuker, M. et al.: Optimal computer folding of large RNA sequences using thermodynamic and auxiliary information. Nucl. Acids. Res. **9** (1981) 133–148

Efficient Initial Pool Generation for Weighted Graph Problems Using Parallel Overlap Assembly

Ji Youn Lee[1], Hee-Woong Lim[2], Suk-In Yoo[2], Byoung-Tak Zhang[2], and Tai Hyun Park[1]

[1] School of Chemical Engineering
[2] School of Computer Science and Engineering,Seoul National University,
San 56-1 Shilim-Dong, Kwanak-Gu, Seoul 151-744, Korea
elfin94@snu.ac.kr, {hwlim, siyoo, btzhang}@bi.snu.ac.kr,
thpark@plaza.snu.ac.kr

Abstract. Most DNA computing algorithms for mathematical problems start with combinatorial generation of an initial pool. Several methods for initial-pool generation have been proposed, including hybridization/ligation and mix/split methods. Here, we implement and compare parallel overlap assembly with the hybridization/ligation method. We applied these methods to the molecular algorithm to solve an instance of the graph problem with weighted edges. Our experimental results show that parallel overlap assembly is a better choice in terms of generation speed and material consumption than the hybridization/ligation method. Simulation of parallel overlap assembly was performed to investigate the potential and the limitation of the method.

1 Introduction

DNA computing has showed its potential by solving several mathematical problems, such as graph and satisfiability problems [1-5]. To solve those problems, precedent initial pool generation is required even though it has been pointed out as a shortcoming in DNA computing. Most molecular algorithms generate initial pools in the first implementation step and then filter the candidate solutions which satisfy the given conditions. Usually, an initial pool is a combinatorial library that contains numerical or indicative information. There are a few initial pool generation methods with their own advantages and disadvantages. One of them is the hybridization/ligation method that link oligonucleotides hybridized through hydrogen bonds by enzymatic reaction. This method was first introduced by Adleman to solve a Hamiltonian path problem [1] and a traveling salesman problem (TSP) [3]. Parallel overlap assembly (POA) was originally introduced by Stemmer to facilitate in vitro mutagenesis [6]. It was successfully applied by Kaplan *et al.* to generate an initial pool consists of binary numbers to solve a maximal clique problem [7]. Other method, such as the mix/split method was introduced by Faulhammer *et al.* to generate combinatorial library of binary numbers [5]. Braich *et al.* applied this method to generate an initial pool for 20-variable 3-SAT problem [4].

In previous work, we implemented a molecular algorithm to solve a 7-city traveling salesman problem [3]. The molecular algorithm for TSP also contained an initial pool generation step before filtering and readout steps. The hybridization/ligation method was used to generate an initial pool and we succeeded to solve the problem; however, the pool generation efficiency was very low. Therefore, the hybridization/ligation method cannot guarantee a complete pool as the problem size increases, which limits the solvable problem size. We introduced another initial pool generation method which is based on parallel overlap assembly. This method was compared with the former one by looking at the product size distributions. Additionally, a computerized simulation of parallel overlap assembly was performed to support the experimental results.

2 Initial Pool Generation Methods

2.1 Parallel Overlap Assembly

Kaplan *et al.* suggested the construction of computational DNA libraries based on a DNA shuffling method [2, 6]. They succeeded in constructing a complete library of binary numbers from 0 to 2^4-1 to solve the maximal clique problem for a graph with four vertices. Their library consisted of two parts; one is the position string of fixed length and the other is value string (0 or 1) of various lengths. The DNA strands corresponding to the same position string were overlapped during an annealing step in the assembly process while the remaining parts of the DNA strands were extended by dNTPs incorporation by polymerase (represented by the dotted arrows in Fig. 1).

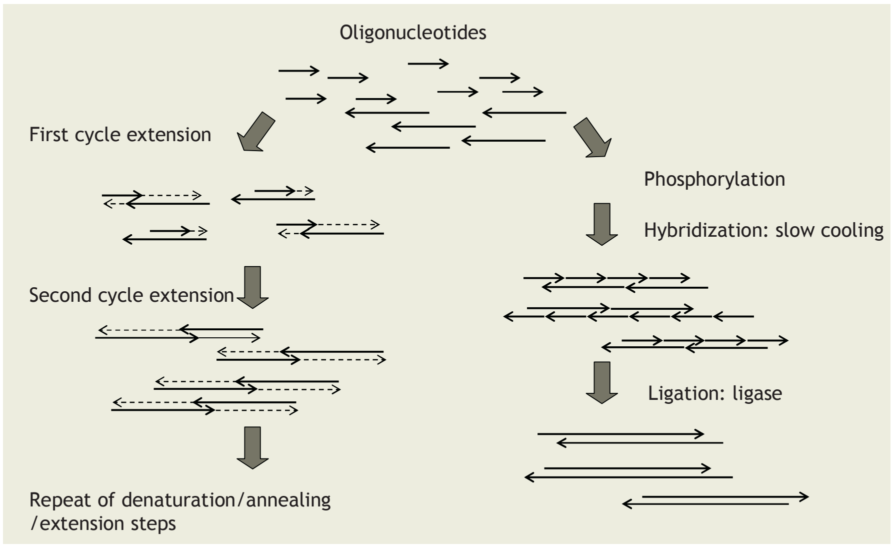


Fig. 1. Schematic diagram of parallel overlap assembly and the hybridization/ligation method for a traveling salesman problem. **Left part**: The thick arrows represent the single-stranded DNA molecules which participate in each cycle of the reaction. The dotted arrows represent the elongated part by dNTPs incorporation. **Right part**: The nicks generated in the hybridization step are linked by ligase via the formation of a phosphodiester bond between the 3' hydroxyl and 5' phosphate of adjacent nucleotides. The arrowhead indicates the 3’ hydroxyl end

The mechanism of POA resembles that of polymerase chain reaction (PCR) in that it repeats the denaturation, annealing and extension. However, the characteristics are complete different. PCR is an in vitro DNA amplification method, so the number of target DNA strands doubled every cycle. In POA, the number of DNA strands does not increase as the cycle progresses, while the lengths of the DNA strands increase.

POA can also efficiently be applied to initial pool generation for a weighted graph problem because the encoding scheme relies on hybridization between city sequences which correspond to the positioning string in the maximal clique problem. The schematic diagram of POA for a traveling salesman problem is shown in Fig. 1.

2.2 Hybridization and Ligation Method

Adleman created an initial pool of candidate paths in parallel to solve a 7-node Hamiltonian path problem utilizing the hybridization/ligation method [1]. Possible paths of various lengths were generated by hybridization between half sequences of each node. Ligase connected the nicks between the 3' hydroxyl and 5' phosphate of adjacent nucleotides which are formed after hybridization via a phosphodiester bond and consequently linked the DNA molecules (right part of Fig. 1). This method was also applied to solve a 7-city traveling salesman problem [3].

3 Materials and Methods

3.1 Target Problem

The target problem was a 7-city traveling salesman problem as shown in Fig. 2 (A). DNA strands representing the city and the cost were encoded with 20-mer oligonucleotides. DNA strands representing the road were encoded according to city and cost information with 40-mer oligonucleotides. The last half (10-mer) of the departure city and the first half (10-mer) of the arrival city act as linkers to connect the cities (Fig. 2 (B)). The linker parts hybridize in hybridization and annealing step in each initial pool generation method.

3.2 Parallel Overlap Assembly

Thirty five different DNA oligonucleotides (7 cities, 5 costs, and 23 roads) were mixed and subjected to PCR without primers as templates. The reaction mixture contained 1.25 unit of Pyrobest® DNA polymerase (TaKaRa, Japan) in 10 mM Tris-HCl, pH 9.0, 1 mM $MgCl_2$, 50 mM KCl, and 0.2 mM of each dNTP was dissolved in distilled water. The total reaction volume was 20 µl. PCR was processed for 34 cycles at 95°C for 30 seconds, at 55°C for 30 seconds and at 72°C for 30 seconds. Initial denaturation and prolonged polymerization were executed for 4 minutes each.

3.3 Hybridization and Ligation

The same amount of oligonucleotide mixture as in POA was prepared [3]. The mixture was heated to 95°C and then hybridized by slow cooling to 20°C at 1°C per minute. The reaction mixture was then subjected to a ligation. For a ligation, 5 µl of

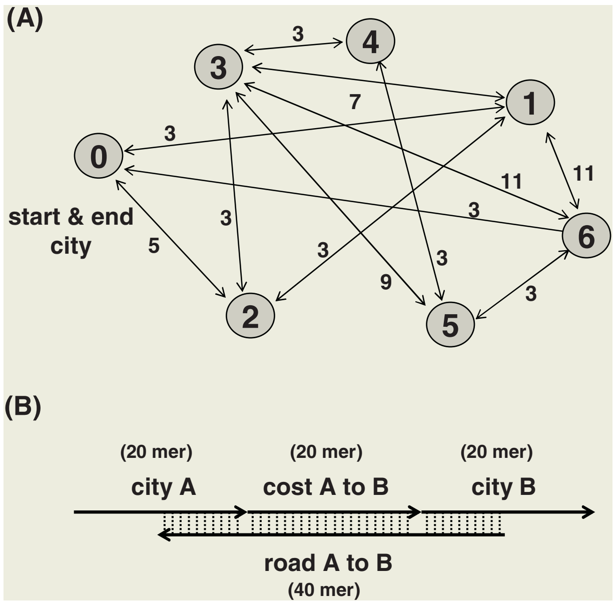


Fig. 2. The seven-city traveling salesman problem (A) and encoding scheme (B). Paths start and end at city 0. The circles denote the cities and the arrows represent the roads. The number on each arrow gives the cost on the given road. The arrowhead of (B) denotes 3' hydroxyl end

the reaction mixtures, 700 units of *T4* DNA ligase (TaKaRa, Japan), ligase buffer (66 mM Tris-HCl, pH 7.6, 6.6 mM $MgCl_2$, 10 mM DTT, 0.1 mM ATP), and an appropriate volume of distilled water was mixed. The total reaction volume was 10 μl. The reaction mixture was incubated at 16°C for 16 hours.

3.4 Gel Electrophoresis and Image Analysis

Agarose gel electrophoresis was performed with 2% Agarose-1000 (Invitrogen, CA, USA) in 0.5X tris-Borate-EDTA buffer and gel was stained with ethidium bromide. As a marker, GeneRuler^{TM} 50 bp DNA ladder (Fermentas, MD, USA) was used. The gel image was obtained with a Gel-Doc and analyzed by Quantity One^{TM} (Bio-Rad, USA).

3.5 Simulation of Parallel Overlap Assembly

We used the algorithm of Maheshri [8] to simulate parallel overlap assembly process for initial pool generation. For simplicity, we only considered match regions to calculate the free energy and did not consider mismatches or dangling ends. The free energy was calculated from the nearest-neighbor model and the parameters given by SantaLucia [9].

4 Results and Discussion

To compare the initial pool generation efficiencies between the hybridization/ligation method and parallel overlap assembly, we performed agarose gel electrophoresis and analyzed the gel image with an image analysis software. The efficiencies can be indirectly compared with the produced amounts of expected length of DNA strands. From the agarose gel, it was possible to determine the length distribution of DNA strands. The length of the candidate paths was 300 bp. The candidate paths contained eight cities and seven costs which were 20-mer respectively; the paths start from city 0, end with city 0, and visit all seven cities. The electrophoresis results are shown in Fig. 3 (A). After the hybridization/ligation reaction, the elongated DNA strands were observed (lane 2 in Fig. 3 (A)) to be located higher than the oligomer mixture (lane 1 in Fig. 3 (A)). The fluorescence intensity was generally increased by the double-stranded DNA formation when compared with oligomer mixture in lane 1. However, most DNA strands are located around 100 bp, which indicates that the generated paths visit only two to four cities. Approximately 13.18 ng of DNA strands were located

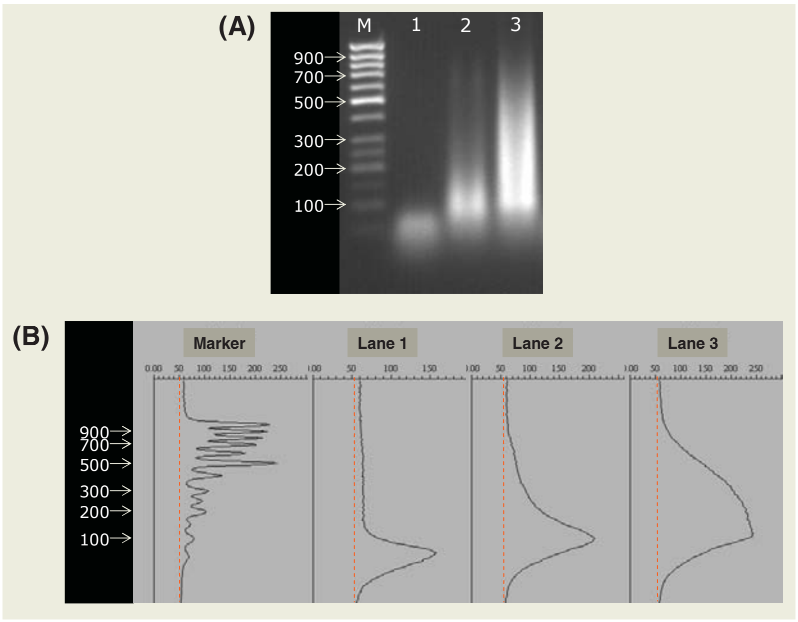


Fig. 3. Comparison of two initial pool generation methods. (A) Experimental results of electrophoresis on 2% agarose gel. M denotes DNA size marker (50 bp ladder). Lane 1 is the oligomer mixture, lane 2 is the product of hybridization/ligation reaction and lane 3 is the product of parallel overlap assembly by *Taq* polymerase. (B) Image analysis results using Quantity One^{TM}. Each graph corresponds to each lane of (A). When comparing POA with the hybridization/ligation method, more DNA strands are located around 300 bp which is the expected pool size

around 300 bp, which corresponds to 67.59 fmole molecules or 27.04 nM (quantified by a linear regression of marker DNA molecules). When considering the reactionvolume (10 µl), the generated pool size was 1.63×10^{11}. The complete initial pool size of the 7-city TSP cannot exceed 8^8 (= 1.68×10^7). Therefore, we can conclude that the initial pool generated by hybridization/ligation contained the complete pool.

In the case of parallel overlap assembly, many longer DNA strands were observed compared to the hybridization/ligation reaction (lane 3 in Fig. 3 (A)). Moreover, the product amount that was represented by the area of the peak was increased by the dNTPs incorporation by polymerase. There were approximately 25.82 ng of DNA strands around 300 bp, which corresponds to 132.41 fmoles or 26.48 nM. When considering the reaction volume (20 µl), the generated pool size was 3.19×10^{11}. The initial pool size generated from the same amount of initial oligonucleotides was about two times larger than that of hybridization/ligation. With larger problem, the initial pool size is too small to contain the complete pool; however, POA with more cycle and large experimental scale can include practical pools.

Parallel overlap assembly is a better initial pool generation method for problems that require combinatorial initial pools, such as weighted graph problems. Firstly, POA is more efficient than the hybridization/ligation method in that it maintains the population size, *i.e.* the number of DNA molecules throughout the procedure. Initially, two single-stranded DNA molecules partially hybridize in the annealing step and then they are extended by dNTPs incorporation by polymerase. The elongated DNA molecules are denatured to two single-stranded DNA in the next denaturation step, and they are subjected to the annealing reaction at the next cycle. Therefore, the population size does not change, and we can decide the population size by varying the initial number of oligonucleotides. On the other hand, in the hybridization/ligation method, the population size decreases as reaction progresses. For example, in our target problem, one complete double-stranded DNA strand composed of eight cities can be made by a ligation of eight city strands, seven cost strands, and seven road strands. This means that the population size is decreased by a factor of the number of components composing it in the hybridization/ligation method. As the problem size increase, the required initial pool size increases dramatically. Therefore, in the light of scalability, POA has an advantage over the hybridization/ligation method.

Secondly, POA does not require phosphorylation of oligonucleotides which is prerequisite for the ligation of oligonucleotides. We used 5'-phosphate group modified oligonucleotides for ligation and the oligonucleotide synthesis cost take up most of the expenses. Thirdly, POA demands less time than the hybridization/ligation method. Hybridization required one and half hour while ligation required more than 12 hours; however POA for 34 cycles required only two hours. Therefore, POA is a much more efficient and economic method for initial pool generation. Moreover, initial pool generation by POA requires fewer strands than the hybridization/ligation method to obtain a similar amount of initial pool DNA molecules, because complementary strands are automatically extended by polymerase. For example, in the above target problem, the cost strands are not required because the cost regions can be filled by polymerase extension.

However, POA was not as efficient as we expected. So, we performed a computerized simulation to investigate the capability and the limitation of POA as an initial pool generation method. In the simulation, each cycle consisted of three steps: two single-stranded DNA molecules were randomly selected and collided; annealing event and duplex formation were decided based on the thermodynamic properties; and extendable duplexes were extended according to the pre-determined polymerase fidelity. The selection probability was proportional to the concentration of each DNA strand. Whether single-stranded DNA molecules will hybridize or not was determined by Boltzman-weighted probability.

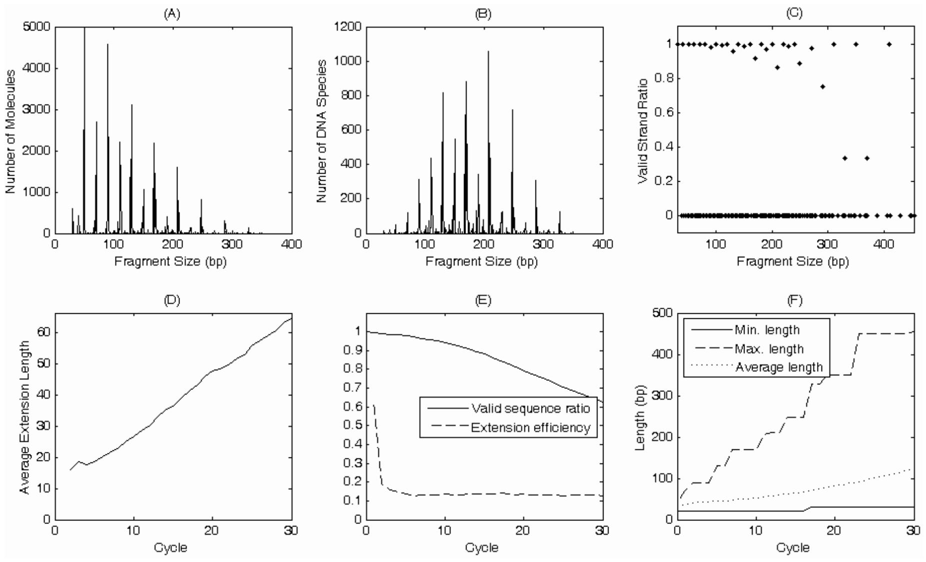


Fig. 4. Simulation results. (A) Length distribution of POA product. (B) Diversity of generated DNA species. (C) Valid strand ratio of final product in each length. (D) Average extension length in each cycle. (E) Valid sequence ratio and extension efficiency in each cycle. (F) Length change during POA: minimum, maximum, and average length in population

After 30 cycles of POA process, various sizes of double-stranded DNA strands were generated (Fig. 4 (A)). The average extension length continuously increased with each cycle (Fig. 4 (D)), which is because extended DNA strands were used as templates for longer strand generation. However, the ratio of DNA strands, which were long enough to contain optimal solution, was very low. This was mainly due to the rapid decrease of the extension efficiency. When we defined the extension efficiency of each cycle as the ratio of annealing event which forms extendable duplex to total annealing event, the extension efficiency dropped dramatically after only a few cycles (Fig 4 (E)). The reason is that non-extendable annealing event increased with cycle. In the early cycles of POA, most annealing between single-stranded DNA molecules formed an extendable duplex. Extendable duplex means that a 3'-end part of one single-stranded DNA molecule is annealed to its complementary part of the other single-stranded DNA molecule, which can undergo extension by polymerase. However, non-extendable duplex formation rapidly increased with each cycle. Non-extendable duplex means that their 3'-end parts are both dangling, so polymerase cannot incorporate dNTP molecules. The probability of annealing event with a dangling 3'-end increased with each cycle, because the elongated DNA strands had long subsequence complementary to the other strand. They underwent re-annealing rather than initiated an extension reaction. This explained the rapid decrease of the extension efficiency.

Products of POA were diverse both in length and in composition as shown in (Fig 4 (B)). However, the valid strand ratio decreased with each cycle, because the DNA strands extended by mis-hybridization were not eliminated or recovered during the process. Elongated DNA strands can be considered as concatenation of the predefined DNA blocks: left half and right half of city strands, cost strands, and their complementary strands (the region connected by dotted lines in Fig. 2 (B)). Valid strands must be the concatenation of the above DNA blocks. However, invalid strands which are extended after mis-hybridization must contain incomplete subsequences of DNA blocks, and this cannot be recovered during the POA process. Moreover, as DNA strands are extended and getting longer, the possibility of a non-specific annealing increases. These are the reasons why the valid sequence ratio decreased with every cycle, and the valid strand ratio of longer strands were lower than that of shorter strands as shown in Fig. 4 (C). For example, when we investigated the invalid strand of 47 bp after first cycle, we could observe an initial dimer formation between city5→city3 road sequences, which caused incomplete DNA block of 7 bp. We could find the same block in the middle of invalid strands of 287 bp after 30 cycles. This block was found at different positions among those strands. This means that this type of mis-hybridization also happened in a later cycle, because the extension of DNA strands is unidirectional. Like this example, invalid strands generated by mis-hybridization accumulate during POA cycles. Unlike in PCR, which primer strands exist in excess, template strands behave as primers in POA, therefore elaborate 3'-end sequence design is critical for successful POA.

Though POA is a better method for initial pool generation than hybridization/ligation method as mentioned in previous section, the efficiency of POA is not enough to be applied to initial pool generation. Sequence design for initial strands of POA is an important factor and especially the specificity of 3'-end must be considered carefully to prevent an extension after non-specific annealing. In addition, we have to incorporate POA with another supplementary process such as gel electrophoresis and additional amplification of target length by PCR.

Acknowledgements

This research was supported in part by the Ministry of Commerce, Industry and Energy through MEC project, the Ministry of Education & Human Resources Development under the BK21-IT Program and the Ministry of Science and Technology through the NRL Program. The ICT at Seoul National University provided research facilities for this study.

References

[1] L. Adleman. Molecular computation of solutions to combinatorial problems. Science, 266:1021-1024, 1994.

[2] Q. Ouyang, P. D. Kaplan, S. Liu, and A Libchaber. DNA solution of the maximal clique problem. Science, 278:446-449, 1997.

[3] J. Y. Lee, S. –Y. Shin, T. H. Park, and B. –T. Zhang. Temperature gradient-based DNA computing for graph problems with weighted edges. Lect. Notes. Compt. Sci. 2568:73-84, 2003.

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

[5] D. Faulhammer, A. R. Cukras, R. J. Lipton, and Laura F. Landweber. Molecular computation: RNA solutions to chess problems. Proc. Natl. Acad. Sci. USA. 98: 1385-1389, 2000.

[6] W. Stemmer. DNA shuffling by random fragmentation and reassembly: in vitro recombination for molecular evolution. Proc. Natl. Acad. Sci. USA. 91: 10747-10751, 1994.

[7] P. D. Kaplan, Q. O. Ouyang, D. S. Thaler, and A Libchaber. Parallel overlap assembly for the construction of computational DNA libraries. J. Theor. Biol., 188: 333-341, 1997.

[8] N. Maheshri, and D. V. Schaffer. Computational and experimental analysis of DNA shuffling. Proc. Natl. Acad. Sci. USA. 100: 3071-3076, 2003.

[9] J. SantaLucia, Jr. A unified view of polymer, dumbbell, and oligonucleotide DNA nearest-neighbor thermodynamics. Proc. Natl. Acad. Sci. USA. 95: 1460-1465, 1998

Partial Words for DNA Coding

Peter Leupold[*]

Research Group on Mathematical Linguistics, Rovira i Virgili University,
Pça. Imperial Tàrraco 1, 43005 Tarragona, Catalunya, Spain
klauspeter.leupold@estudiants.urv.es

Abstract. A very basic problem in all DNA computations is finding a good encoding. Apart from the fact that they must provide a solution, the strands involved should not exhibit any undesired behaviour, especially they should not form secondary structures. Various combinatorial properties like repetition-freeness and involution-freeness have been proposed to exclude such misbehaviour. Another option, which has been considered, is requiring a big Hamming distance between the codewords.

We propose to consider partial words for the solution of the coding problem. They, in some sense, already include the Hamming distance in the definition of compatibility and are investigated for many combinatorial properties. Thus, they can be used to guarantee a desired distance and simultaneously other properties. As the investigations on partial words are attracting more and more attention, they might be able to provide an ever-growing toolbox for finding good DNA encodings.

1 Introduction

Partial words were introduced by Berstel and Boasson in 1998 [3]. One of their main motivations came from the behaviour of DNA strands. Two strands complementing each other very closely, but having a few mismatches can still align with each other. In such a double strand of DNA with a few mismatches, one cannot tell which of the two non-matching bases is the right or original one. Thus one might consider the respective position as one without information, a whole, and then see what can still be said about the resulting word.

On the other hand, DNA or RNA strands are the basic building blocks and the carriers of information in all DNA computations. There, a major issue is finding the right words to encode the problem under consideration. Computations like the famous, seminal experiment by Adleman depend on the usage of sequences, which will recombine exactly in the ways intended in the design.

Of course, the central problem is finding a suitable set of linear sequences, whose recombinations will constitute the computation. However, in addition the designer faces further problems due to the fact that in reality nucleic acid strands

[*] This work was done, while the author was funded by the Spanish Ministry of Culture, Education and Sport under the Programa Nacional de Formación de Profesorado Universitario.

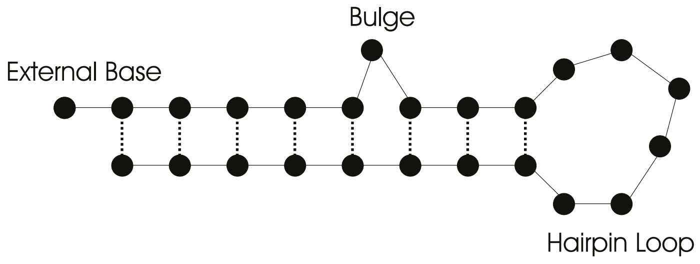


Fig. 1. Various possible secondary structures of an RNA strand; the line indicates the backbone, the dotted ones paired bases

are not linear but three-dimensional objects. Thus the selected strands might form three-dimensional secondary structures like loops and hairpins. This should be avoided by all means, i.e. no parts of the strand should align with other parts of the same strand to form structures like the ones depicted in Figure 1. Such secondary structures obviously hinder further combinability, readability etc., and thus can render useless a computation designed very well for one-dimensional strings, but without consideration of the actual behaviour of longer strands of nucleic acids in the three-dimensional real world.

Adronescu et al. also mention a rather simple model trying to predict such structures, the *no repeated k-strings* model [1]. There, a sequence is supposed to have non-empty secondary structure, if there is a repetition without overlap of a factor of length k in the string. We want to emphasize that this need not be a direct repetition, usually termed square in combinatorics on words – in this case an arbitrarily long sequence can separate the two repetitions. It should be added that in an application one would really be looking for *reverse complement* strings instead of repetitions. This, however, could be done with essentially the same algorithms.

In a different approach Deaton et al. have considered the Hamming distance between the strands of a coding as an approximate measure for the reliability of a computation [9]. The farther apart the code words are, the less probable undesired bindings are.

Partial words seem a good tool to combine properties like these two: if we replace equality of words by compatibility, we in some sense get the Hamming distance for free. If then we define a property like repetition-freeness also for partial words, we can guarantee both properties in a unified way.

We will proceed to illustrate this considering another property, namely involution-freeness, which was introduced by Hussini et al. [13] and further developed with numerous variants in subsequent work [11, 12, 14]. Before this, we mention yet another motivation to use partial words in the context of DNA computation: if such a computation lasts for a longer time and involves some recombinations and especially copying of strands, errors are bound to be introduced in some of the strands; for example, copying processes never work with absolute perfection. If one wants to choose a set of DNA words fulfilling a property, even after some bases may have been changed, it is an appropriate approach to check the following: does the desired property still hold, if the originally chosen language of code words and their possible catenations is punctured up to a certain degree with holes like in a partial word. Such a code language would be more robust to errors and external influence and would thus promise more reliable computations. Also the production of not exactly complementary but very similar sequences, which still might align to each other, would be outruled.

2 Partial Words

The main motivation for the introduction of partial words mentioned by Berstel and Boasson [3] came from molecular biology of nucleic acids. There, among other things, one tries to determine properties of the DNA or RNA sequences encountered in nature. These are usually seen as strings over the alphabet $\{A, T, C, G\}$, respectively $\{A, U, C, G\}$ of four bases. In nature they mostly occur paired with their Watson-Crick complements; all these concepts belong more to the realm of biology and will not be explained in any depth here.

But supposing only perfect pairings of bases is supposing an ideal world. As long as the number of mismatches is not very high, similar strands will still align due to the affinity of their matching parts.

If one then encounters a pair of strands as depicted in Figure 2, there is in general no telling, which one of the mismatched bases is the correct or original one.

However, one still wants to investigate the properties of such a sequence and state them as concisely as possible. To this end, it seems a plausible choice to regard the positions in question as unknown, or holes, and to see what then still can be said about such a sequence. In the given example we would consider (for the upper strand) a string composed of the parts $\ldots AG$, $CAAUGU$, and $ACAGUC\ldots$ in this order with one hole inbetween each of the parts.

Thus, intuitively, a partial word is very much like a conventional word, only at some positions we do not know which letter it has. Looking at a word as a total function from $\{0, \ldots, |w| - 1\}$ to Σ, we then except these unknown positions from the mapping's domain and define a *partial word* w as a partial function from $\{0, \ldots, |w| - 1\}$ to Σ. The positions, where $w[n]$ is not defined for $n < |w|$ are called the word's *holes*. The numbers in $\{0, \ldots, |w| - 1\} \setminus D(w)$ are the set of holes of w and are written $\mathsf{Hole}(w)$. Here $D(w)$ denotes the domain of w.

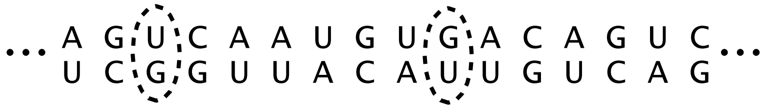

... A G U C A A U G U G A C A G U C ...
U C G G U U A C A U U G U C A G

Fig. 2. Part of an RNA sequence with two mismatches

For a partial word w we define its *companion* as the total word $w_\Diamond$ over the extended alphabet $\Sigma \cup \{\Diamond\}$ where

$$w_\Diamond[i] := \begin{cases} w[i] & \text{if } i \in D(w) \\ \Diamond & \text{if } i \notin D(w) \wedge 0 \leq i < |w| \end{cases}$$

When it is more convenient, we will also refer to the companion as a partial word to simplify the syntax of our sentences. Thus we will say for example "the partial word $\Diamond a \Diamond b$" instead of "the partial word with companion $\Diamond a \Diamond b$".

For two partial words u and v of equal length, we say that u is *contained* in v, if $D(u) \subset D(v)$ and $i \in D(u) \rightarrow u[i] = v[i]$; this is written $u \subset v$, a rather natural notation, if we adopt the view of a function f as a set of ordered pairs $[n, f(n)]$. If there exists a partial word w such that for two other partial words u and v we have $u \subset w$ and $v \subset w$, then u and v are called *compatible*, written $u \uparrow v$. For two such words, $u \vee v$ is the smallest word containing both u and v; smallest here means that its domain is $D(u \vee v) = D(u) \cup D(v)$, its values are defined in the obvious way.

At times it will be interesting to in some sense measure to what degree a partial word is riddled with holes. For example a nucleic acid sequence would certainly not align with its complement any more, if more than half of its bases had been changed. To formally denote the degree to which a partial word u is undefined we will use the *puncturedness coefficient* defined as $\vartheta(u) := \frac{\mathsf{Hole}(u)}{|u|}$.

A final notion we will need concerns the intersection of two sets of partial words. For it to contain also words not in either language but compatible to at least one word from each language, we define a modified intersection:

$$K \sqcap L := \{w : \exists u \in K, \exists v \in L[w = u \vee v]\}$$

An important question is how to obtain such languages of partial words to start with. Because, in this context, holes are considered as some type of defect, which might occur just about anywhere, we choose the following approach: we start out from a language L of total words.

Definition 2.1 *For some puncturedness coefficient r with $0 < r \leq 1$ the language $L^{r-\Diamond}$, called L's* r-puncturing, *is the one that contains all words of L and all the partial words one can obtain from these obeying the bound imposed by the coefficient r.*

As already mentioned, the Hamming distance between code words has been considered as a measure for the quality of a code [9]. To make evident the close connection between compatibility and the Hamming distance we close this section by stating a rather obvious equivalence.

Proposition 2.1 *Let k, m be two natural numbers with $m > 2k$. Two words $u, v \in (\Sigma^m)^{k-\Diamond}$ are compatible, if and only if their Hamming distance is less than or equal $2k$.*

Notions from classical Formal Language Theory are not explained here; we only mention that Σ shall always denote the alphabet under consideration and that Σ^* is the set of all words over this alphabet; finally $\Sigma^+ := \Sigma^* \setminus \{\lambda\}$.

3 Involutions and DNA

We now introduce a special class of mappings, so-called involutions. These enjoy some special interest in the context of DNA computing, because the Watson-Crick complementarity corresponds to a specific involution to be introduced further down.

In general, an *involution* is a mapping θ such that θ^2 is the identity mapping. A mapping such that always $\mu(uv) = \mu(u)\mu(v)$ is a *morphism*; an involution also fulfilling this property will be called a *morphic involution.*

We now recall a few special involutions acting over the DNA-alphabet $\Delta = \{A, C, G, T\}$, which were introduced, for example, by Hussini et al [11]; their specific importance in the context of DNA is that a strand and its image under τ align. Thus, strands which are not supposed to align with themselves to form secondary structures should not contain at the same time some factor and its image as explained with more detail in the cited source.

The complement involution γ is defined by $\gamma(A) := T$, $\gamma(T) := A$, $\gamma(G) := C$, and $\gamma(C) := G$; additionally we define $\gamma(\Diamond) := \Diamond$ to extend the mapping from total to partial words. A second involution is the mirror involution μ mapping every word into its mirror image; i.e. it reverses the word's order. Their combination $\mu\gamma$ is also an involution and will be called the DNA involution τ. Thus, for example $\tau(GTAT) = ATAC$.

4 Involution Compliance and Freedom

For various reasons it seems reasonable to consider in this context only puncturedness bounds relative to the words' length. First, whether two strands align or not, depends not on the absolute number of mismatches, but more on their frequency. While two strands of length eight will almost certainly not align, if there are four mismatches, the same number is negligible for strands of lengths greater than one hundred.

Secondly, computation means that something is happening; so computing with DNA molecules means that they are changed, at the very least rearranged in some way. Often enough this involves catenation; especially in the matters treated here, catenation plays a central role. And while relative bounds are preserved under catenation, absolute ones are not, because the number of holes is simply summed up for two catenated words.

Proposition 4.1 *For a rational number* r *with* $0 < r < 1$ *the inclusion* $(L^{r-\Diamond})^* \subseteq (L^*)^{r-\Diamond}$ *holds.*

It is easily seen that the two sets are in general not equal. Consider a non-empty language L with only words shorter than $\frac{1}{r}$; its r-puncturing is just L itself, it does not contain any words with holes. Thus also the Kleene iteration is a total language. The r-puncturing of L^*, however, contains words of arbitrary length and therefore also words with holes.

To make notation a little easier and more readable we make the following convention: when the puncturing symbol $^\Diamond$ will be used without giving either a constant or a relative bound in the form $^{r-\Diamond}$; this shall mean that all occurences within a definition, theorem, etc. have the same relative bound. In general, the respective statements will not be true or make sense without this unstated assumption.

Definition 4.1 A language of partial words L is θ-compliant for a morphic involution θ, if for words $u, v, w, w' \in \Sigma^{*\Diamond}$ we have that $w, u\theta(w')v \in L^\Diamond$ and $w \uparrow \theta(w')$ imply $uv = \lambda$; if also $(L)^\Diamond \sqcap \theta(L)^\Diamond = \emptyset$, then L is strictly θ-compliant.

A rather easy to see property of compliance is the following.

Proposition 4.2 *For every θ-compliant language $L^\Diamond$, also $\theta(L^\Diamond)$ is θ-compliant.*

Further, we can see quickly a necessary and sufficient condition for the strictness of compliance.

Proposition 4.3 *A θ-compliant language $L^\Diamond$ is strictly θ-compliant, if and only if $(L^*)^\Diamond \sqcap (\theta(L)^*)^\Diamond = \{\lambda\}$.*

Proof. The empty word is in any iteration of a language, and thus always $\lambda \in (L^\Diamond)^* \sqcap (\theta(L)^\Diamond)^*$. Now suppose there is another word in this set for some strictly θ-compliant language L. This means there are words $u_1, \ldots, u_n$ and $v_1, \ldots, v_m$ all from L such that

$$(u_1 \cdot u_2 \ldots u_n)^\Diamond \uparrow \theta(v_1 \cdot v_2 \ldots v_m)^\Diamond.$$

If $|u_1| < |\theta(v_1)|$ or $|u_1| > |\theta(v_1)|$ this leads to a contradiction to L's θ-compliance or to $\theta(L)$'s θ-compliance, which by Proposition 4.2 is equivalent. For $|u_1| = |\theta(v_1)|$ we must have $u_1 \uparrow \theta(v_1)$; so only in this case we need the *strict* θ-compliance of L to reach a contradiction.

The other direction of the implication is rather obvious. □

After compliance, freeness is a second interesting property related to involutions.

Definition 4.2 A language L is θ-free, if $(L^2)^\Diamond \sqcap (\Sigma^+\theta(L)\Sigma^+)^\Diamond = \emptyset$; if also $(L)^\Diamond \sqcap \theta(L)^\Diamond = \emptyset$, then L is strictly θ-free; it suffices, if $L \setminus \{\lambda\}$ is strictly θ-free.

It is quite clear that the notions just defined carry over from punctured languages to total ones, in exactly the sense of the original definitions [11]. We state this explicitly only in one exemplary case.

Proposition 4.4 *If any puncturing of a language L is θ-free, then L itself is θ-free.*

Proof. For languages of only total words, $\sqcap$ becomes simply conventional set intersection; thus the two definitions of θ-freeness are equivalent, and consequently hold for the same class of languages, □

As is to be expected, the contrary is not true. To show this, we investigate an example provided by Hussini et al. Here the DNA-involution τ and the DNA-alphabet Δ are used as introduced in Section 3.

Example 4.1 $ACC\Delta^2$ is τ-free [11]. With a puncturing factor of $\frac{1}{10}$ this is not true any more. Consider the two words

$$\begin{aligned} w_1 &= \texttt{ACCGGACCTG} \\ w_2 &= \texttt{ACCGGTCCTG}, \end{aligned}$$

where w_1 is from $ACC\Delta^2 ACC\Delta^2$, and $\tau(w_2)$ is from the set $\Delta^+(\Delta^2 GGT)\Delta^+$, which is equal to $\Delta^+\theta(ACC\Delta^2)\Delta^+$. They are identical except for the sixth position. As they have length ten, one hole is allowed, and thus the $\frac{1}{10}$-puncturing of $ACC\Delta^2$ is not τ-free, because $(ACC\Delta^2 ACC\Delta^2)^\Diamond \sqcap (\Delta^+(\Delta^2 GGT)\Delta^+)^\Diamond$ contains, for example, $\texttt{ACCGG}\Diamond\texttt{CCTG}$, because this word is contained in both languages.

So we see that in general things must be reinvestigated. However, we have chosen our definitions in a way that leaves most results for total languages valid with only the reformulations necessary by the slight differences in definition. We illustrate this with the first result from Hussini et al.

Proposition 4.5 *For a language L and a morphic involution θ the following hold true:*
(i) If $L^\Diamond$ is θ-free, then both $L^\Diamond$ and $\theta(L^\Diamond)$ are θ-compliant.
(ii) If $L^\Diamond$ is strictly θ-free, then $(L^2)^\Diamond \sqcap (\Sigma^*\theta(L)\Sigma^*)^\Diamond = \emptyset$

Proof. We give only the proof for (i); it is very analogous to the proof of the original lemma just as the proof for part (ii). So suppose that L is a language such that $L^\Diamond$ is θ-free but at the same time not θ-compliant. The latter implies that either $\Sigma^+\theta(L^\Diamond)\Sigma^* \sqcap L^\Diamond \neq \emptyset$ or $\Sigma^*\theta(L^\Diamond)\Sigma^+ \sqcap L^\Diamond \neq \emptyset$. Catenating $L^\Diamond$ on the right side in the second case, we obtain $L^\Diamond\Sigma^*L^\Diamond\Sigma^+ \sqcap L^\Diamond L^\Diamond \neq \emptyset$. With Proposition 4.1 we see that then also $(L\Sigma^*L\Sigma^+)^\Diamond \sqcap (LL)^\Diamond \neq \emptyset$; this contradicts θ-freeness. In the first case an analogous contradiction is reached. Finally, for $\theta(L^\Diamond)$ the inclusion now follows with Proposition 4.2. □

As already mentioned, there is some special interest in the behaviour of certain properties with respect to catenation. For the case of compliance, we can state a positive result in this respect.

Proposition 4.6 *If two languages $L_1^\Diamond$ and $L_2^\Diamond$ are both θ-compliant a morphic involution θ, then also their catenation $L_1 \cdot L_2$ is θ-compliant.*

Proof. We assume that there exist two θ-compliant languages L_1 and L_2, whose catenation is not θ-compliant. This means there are partial words u, v, w, w' such that $w, u\theta(w')v \in L_1^\Diamond \cdot L_2^\Diamond$ and $w \uparrow \theta(w')$. Let w be composed from $w_1 \in L_1^\Diamond$ and $w_2 \in L_2^\Diamond$, and $u\theta(w')v$ from $z_1 \in L_1^\Diamond$ and $z_2 \in L_2^\Diamond$. The border between z_1 and z_2 must be inside the factor $\theta(w')$; otherwise we obtain an immediate contradiction to the θ-compliance of either $L_1^\Diamond$ or $L_2^\Diamond$ looking at w_1 respectively w_2.

So w' has a factorization $w'_1 w'_2$ such that $u\theta(w_1) \in L_1^\Diamond$ and $\theta(w_2)v \in L_2^\Diamond$. Now if $|w_1| \le |w'_1|$, then w_1 is compatible to a prefix of $\theta(w'_1)$ in the word $u\theta(w'_1)$, which is in contradiction to the θ-compliance of $L_1^\Diamond$, because $u\theta(w_1) \in L_1^\Diamond$. If, on the other hand, $|w_1| > |w'_1|$, then $|w_2| \le |w'_2|$; we obtain an analogous contradiction to the θ-compliance of $L_2^\Diamond$. □

5 Constant Length Codings

Now we will restrict our attention to a class of languages, which seems of special interest in the context of DNA computations. Many times all the original strands employed at the beginning of an experiment have the same length. Among other advantages this allows, for example, telling how far the experiment has proceeded in a simple way: the length of the present strands corresponds directly to the number of catenations, which created them. And determining strand length via gel electrophoresis is a very reliable standard procedure.

Thus it seems reasonable to consider languages all of whose words have equal length. This property allows the statement of some results, which are not true in general. We note further that the DNA involution τ as well as its components μ and γ all are length-preserving (this means the length of original and image is always the same); therefore this is a sensible restriction to put on the involutions under consideration.

First off, we note that strictness loses its meaning for involution compliance.

Proposition 5.1 *A language $L \subset \Sigma^{n\Diamond}$ is θ-compliant for a length-preserving involution θ, if and only if it is strictly θ-compliant.*

Proof. Immediate from the definition: For non-θ-compliance $u\theta(w')v$ (variable names referring to Definition 4.1) must have the same length as w. Because always $|w| = |w'| = |\theta(w')|$ for $w, w' \in L$, uv is empty in any counterexample to θ-compliance. □

For involution-freeness, the analogous statement is not true as shown by the following example.

Example 5.1 The set $\{TGGT, ATAC\}$ is strictly τ-free. If we add the word $ACCA = \tau(TGGT)$, the resulting set is still τ-free, the strictness, however, is lost.

Proposition 5.2 *A constant-length language $L^\Diamond$ is strictly θ-free for a length-preserving involution θ, if and only if it is θ-free and θ-compliant.*

Proof. From strict θ-freeness, θ-freeness follows by definition; θ-compliance follows by Proposition 4.5.

The inverse inclusion follows immediately from Proposition 5.1 and the fact that the strictness condition is the same for both freeness and compliance. □

Now we provide a sufficient condition for a constant-length language to be θ-free, and also for strict θ-freeness. Here $\mathsf{pref}(L)$ denotes the set of all proper prefixes of words in L and $\mathsf{suff}(L)$ denotes the set of all proper suffixes including the empty word. For one word the sets of prefixes and suffixes are both finite. Therefore the conditions provided can be checked very easily for any finite language.

However, here we need to give the puncturedness bounds explicitly, because they are not uniform for all languages involved – there is even a mixture of relative and absolute bounds. The proof will make clear, why this is necessary.

Proposition 5.3 *For $L \subset \Sigma^n$, the language $L^{r-\Diamond}$ is θ-free, if and only if $(\mathit{pref}(L)\mathit{suff}(L))^{k-\Diamond} \sqcap L^{r-\Diamond} = \emptyset$ for $k := \lfloor 2 \cdot r \cdot n \rfloor$.*

Proof. If $L^{r-\Diamond}$ is not θ-free, then there exist total words $u, v, w \in L$, such that a word from $\theta(w^{r-\Diamond})$ is compatible to a subword of one from $(uv)^{r-\Diamond}$. This subword is neither entirely from $u^{2r-\Diamond}$ nor entirely from $v^{2r-\Diamond}$ due to the definition of θ-freeness. Further, because all three words have equal length the subword touches both factors.

Since this subword is shorter than uv itself, the local puncturedness coefficient might be higher than the one for the entire word. The maximum number of holes in this word is the following: the length of uv is $2n$, thus with a puncturedness coefficient of r there can be at most $\lfloor 2 \cdot r \cdot n \rfloor$ holes in the entire word from $(uv)^{r-\Diamond}$ – in the extreme case all of them can be in the subword considered above. Thus the set $(\mathsf{pref}(L)\mathsf{suff}(L))^{k-\Diamond} \sqcap L^{r-\Diamond}$ is non-empty. After these considerations also the inverse inclusion follows easily. □

By extending the prefix and suffix sets by the original words themselves, we immediately obtain an analogous characterization of strictly θ-free languages.

Corollary 5.3.a *For $L \subset \Sigma^n$, the language $L^{r-\Diamond}$ is strictly θ-free, if and only if $((L \cup \mathit{pref}(L))^{2r-\Diamond}(L \cup \mathit{suff}(L))^{2r-\Diamond}) \sqcap L^{r-\Diamond} = \emptyset$.*

6 Outlook

What was presented here is only one example for possible usage of partial words in the context of biological computation, or more general, in dealing with DNA sequences with certain desired or undesired properties. In some contexts also a variation of partial words might be useful.

As suggested by both Gh. Păun and G. Lischke, regarding a base pair like $A - G$ as a hole is to some extent giving away some more information than necessary. Although theoretically possible, it is in practice improbable that both bases have been produced by an error. So instead of regarding the position as a complete unknown, one might attribute to it a type like $\{A{-}T, C{-}G\}$, supposing that only the A or only the G is wrong. Considering only one strand we would use $\{A, C\}$; either way, this should then be treated as a letter compatible to all its elements, but not to the other letters.

For the DNA alphabet Δ, the version extended in this sense would be

$$\{A, C, G, T, \{A, C\}, \{A, G\}, \{T, C\}, \{T, G\}\}.$$

The other possible binary combinations like $\{A, T\}$ are, of course well-matched pairs and do not appear here explicitly, but are already represented by the single letters. To our knowledge this type of partiality has not been investigated yet.

We have stated before that using partial words in the original form for the encoding of a DNA computation is very related to the use of Hamming distances between the code words. Already now combinatorial investigations on partial words offer quite a number of tools concerning periodicity [7],[17], primitivity [5], codes [6],[15] etc. Thus, and as the combinatorial theory around partial words is growing, their use might have the advantage over the plain Hamming distance that many properties have already been investigated and may provide ways to guarantee some desired properties. Of course, taking into account the peculiarities of DNA, also some tailor-made restrictions of partial words might be defined and investigated for special purposes.

Acknowledgement

The author is thankful to Max Garzon for pointing him to the coding problem. Further thanks are due to an anonymous referee for close reading of and detailed comments about the manuscript.

References

1. M. Andronescu, D. Dees, L. Slaybaugh, Y. Zhao, A.E. Condon, B. Cohen, and S. Skiena: *Algorithms for Testing That Sets of DNA Words Concatenate without Secondary Structure.* In: [10], pp. 182–195.
2. W. Bauer, H. Ehrig, J. Karhumäki and A. Salomaa (eds.): *Formal and Natural Computing.* Lecture Notes in Computer Science 2300, Springer-Verlag, Berlin, 2002.
3. J. Berstel and L. Boasson: *Partial Words and a Theorem of Fine and Wilf.* In: Theoretical Computer Science, Vol. 218, 1999, pp. 135–141.
4. J. Berstel and D. Perrin: *Theory of Codes.* Academic Press, 1985.
5. F. Blanchet-Sadri: *Primitive Partial Words.* Preprint 2003.
6. F. Blanchet-Sadri: *Codes, Orderings, and Partial Words.* Preprint 2003.
7. F. Blanchet-Sadri and A. Hegstrom: *Partial Words and a Theorem of Fine and Wilf Revisited.* In: Theoretical Computer Science, Vol. 270, No. 1/2, 2002, pp. 401–419.
8. J. Chen and J.H. Reif (Eds.): *DNA Computing, 9th International Workshop on DNA Based Computers.* Lecture Notes in Computer Science 2943, Springer-Verlag, Berlin, 2004.
9. R. Deaton, M. Garzon, R.C. Murphy, J.A. Rose, D.R. Franceschetti and S.E. Stevens Jr.: *On the Reliability and Efficiency of a DNA-based Computation.* In: Physical Review Letters 80:2, 1998, pp. 417–420.

10. M. Hagiya and A. Ohuchi (eds.): *DNA Computing — 8th Int. Workshop on DNA-Based Computers.* Lecture Notes in Computer Science 2568, Springer-Verlag, Berlin, 2003.
11. S. Hussini, L. Kari and S. Konstantinidis: *Coding Properties of DNA Languages.* In: Theoretical Computer Science, Vol. 290, 2003, pp. 1557–1579.
12. N. Jonoska and K. Mahalingam: *Languages of DNA Based Code Words.* In: [8], pp. 61-73.
13. L. Kari, R. Kitto and G. Thierrin: *Codes, Involutions and DNA Encodings.* In: [2].
14. L. Kari, S. Konstantinidis, E. Losseva and G. Wozniak: *Sticky-free and Overhang-free DNA Languages.* In: Acta Informatica 40(2), 2003, pp. 119-157.
15. P. Leupold: *Languages of Partial Words.* Submitted.
16. G. Rozenberg and A. Salomaa (eds.): *Handbook of Formal Languages.* Springer-Verlag, Berlin, 1997.
17. A.M. Shur and Y.V. Gamzova: *Periods' Interaction Property for Partial Words.* In: Preproceedings of Words'03, TUCS General Publications, Turku, 2003.
18. H.J. Shyr: *Free Monoids and Languages.* Hon Min Book Company, Taichung, 1991.

Accepting Hybrid Networks of Evolutionary Processors

Maurice Margenstern[1], Victor Mitrana[2], and Mario J. Pérez-Jiménez[3]

[1] LITA, UFR MIM, University of Metz,
Ile du Saulcy, 57045 Metz-Cedex, France
margens@lita.univ-metz.fr

[2] Faculty of Mathematics and Computer Science, University of Bucharest,
Str. Academiei 14, 70109, Bucharest, Romania
and
Research Group in Mathematical Linguistics, Rovira i Virgili University,
Pça. Imperial Tarraco 1, 43005, Tarragona, Spain
vmi@fll.urv.es

[3] Department of Computer Science and Artificial Intelligence,
University of Seville
Mario.Perez@cs.us.es

Abstract. We consider time complexity classes defined on accepting hybrid networks of evolutionary processors (AHNEP) similarly to the classical time complexity classes defined on the standard computing model of Turing machine. By definition, AHNEPs are deterministic. We prove that the classical complexity class **NP** equals the set of languages accepted by AHNEPs in polynomial time.

1 Introduction

The origin of networks of evolutionary processors (NEPs for short) is twofold. In [5] we consider a computing model inspired by the evolution of cell populations, which might model some properties of evolving cell communities at the syntactical level. Cells are represented by words which describe their DNA sequences. Informally, at any moment of time, the evolutionary system is described by a collection of words, where each word represents one cell. Cells belong to species and their community evolves according to mutations and division which are defined by operations on words. Only those cells are accepted as surviving (correct) ones which are represented by a word in a given set of words, called the genotype space of the species. This feature parallels with the natural process of evolution.

On the other hand, a basic architecture for parallel and distributed symbolic processing, related to the Connection Machine [10] as well as the Logic Flow paradigm [6], consists of several processors, each of them being placed in a node of a virtual complete graph, which are able to handle data associated with the respective node. Each node processor acts on the local data in accordance with some predefined rules, and then local data becomes a mobile agent which can navigate in the network following a given protocol. Only such data can be communicated which can pass a filtering process. This filtering process may require to satisfy some conditions imposed by the sending processor, by the receiving processor or by both of them. All the nodes send simultaneously their data and the receiving nodes handle also simultaneously all the arriving messages, according to some strategies, see, e.g., [7, 10].

In [1](further developed in [2, 11, 3]), we modify this concept (considered in [4] from a formal language theory point of view) in the following way inspired from cell biology. Each processor placed in a node is a very simple processor, an evolutionary processor. By an evolutionary processor we mean a processor which is able to perform very simple operations, namely point mutations in a DNA sequence (insertion, deletion or substitution of a pair of nucleotides). More generally, each node may be viewed as a cell having genetic information encoded in DNA sequences which may evolve by local evolutionary events, that is point mutations. Each node is specialized just for one of these evolutionary operations. Furthermore, the data in each node is organized in the form of multisets of words (each word appears in an arbitrarily large number of copies), and all copies are processed in parallel such that all the possible events that can take place do actually take place. Obviously, the computational process described here is not exactly an evolutionary process in the Darwinian sense. But the rewriting operations we have considered might be interpreted as mutations and the filtering process might be viewed as a selection process. Recombination is missing but it was asserted that evolutionary and functional relationships between genes can be captured by taking only local mutations into consideration [12]. Consequently, hybrid networks of evolutionary processors might be viewed as bio-inspired computing models. We want to stress from the very beginning that we are not concerned here with a possible biological implementation, though a matter of great importance.

In a series of papers, we present *linear* time solutions to some NP-complete problems using these simple mechanisms. Such solutions are presented for the Bounded Post Correspondence Problem in [1], for the "3-colorability problem" in [2] (with simplified networks), and for the Common Algorithmic Problem in [11]. In this paper, we consider time complexity classes defined on accepting hybrid networks of evolutionary processors (AHNEP) similarly to the classical time complexity classes defined on the standard computing model of Turing machine. By definition, AHNEPs are deterministic. We prove that **NP** equals the class of languages accepted by AHNEPs in polynomial time.

2 Basic Definitions

We start by summarizing the notions used throughout the paper. An *alphabet* is a finite and nonempty set of symbols. The cardinality of a finite set A is written $card(A)$. Any sequence of symbols from an alphabet V is called *string (word)* over V. The set of all strings over V is denoted by V^* and the empty string is denoted by ε. The length of a string x is denoted by $|x|$ while $alph(x)$ denotes the minimal alphabet W such that $x \in W^*$.

A nondeterministic *Turing machine* is a construct $T = (Q, V, U, \delta, q_0, B, F)$, where Q is a finite set of states, V is the input alphabet, U is the tape alphabet, $V \subset U$, q_0 is the initial state, $B \in U \setminus V$ is the "blank" symbol, $F \subseteq Q$ is the set of final states, and δ is the transition mapping, $\delta : (Q \setminus F) \times U \longrightarrow 2^{Q \times U \times \{R,L\}}$. Moreover, if $(s, B, X) \in \delta(q, a)$ for some $s, q \in Q$ and $X \in \{R, L\}$, then $a = B$, i.e. T never write B over a symbol different than B. The variant of a Turing machine we use in this paper can be described intuitively as follows: it has a tape divided into cells that may store symbols from U (each cell may store exactly one symbol from U). The tape is semi-infinite, namely it is bounded to the left (there is a leftmost cell) and unbounded (arbitrarily long) to the right. The machine has a a central unit which can be in a state from a finite set of states, and a reading/writing tape head which can scan in turn the tape cells. This head cannot go the the left-hand end of the tape. The input word is a word over V and is stored on the tape starting with the leftmost cell and all the other tape cells contain the symbol B.

Initially, the tape head scans the leftmost cell and the central unit is in the state q_0. The machine performs moves. A move depends on the contents of the cell currently scanned by the tape head and the current state of the central unit. A move consists of: change the state, write a symbol from U on the current cell and move the tape head one cell either to the left (provided that the cell scanned was not the leftmost one) or to the right. An input word is *accepted* iff after a finite number of moves the Turing machine enters a final state.

An *instantaneous description* (ID for short) of a Turing machine T as above is a string over $(U \setminus \{B\})^* Q (U \setminus \{B\})^*$. Given an ID $\alpha q \beta$, this means that the tape contents is $\alpha\beta$ followed by an infinite number of cells containing the blank symbol B the current state is q, and the symbol currently scanned by the tape head is the first symbol of β provided that $\beta \neq \varepsilon$, or B, otherwise.

We say that a rule $a \to b$, with $a, b \in V \cup \{\varepsilon\}$ is a *substitution rule* if both a and b are not ε; it is a *deletion rule* if $a \neq \varepsilon$ and $b = \varepsilon$; it is an *insertion rule* if $a = \varepsilon$ and $b \neq \varepsilon$. The set of all substitution, deletion, and insertion rules over an alphabet V are denoted by Sub_V, Del_V, and Ins_V, respectively.

Given a rule as above σ and a string $w \in V^*$, we define the following *actions* of σ on w:

- If $\sigma \equiv a \to b \in Sub_V$, then
$$\sigma^*(w) = \sigma^r(w) = \sigma^l(w) = \begin{cases} \{ubv : \ \exists u, v \in V^* \ (w = uav)\}, \\ \{w\}, \ \text{otherwise} \end{cases}$$
- If $\sigma \equiv a \to \varepsilon \in Del_V$, then $\sigma^*(w) = \begin{cases} \{uv : \ \exists u, v \in V^* \ (w = uav)\}, \\ \{w\}, \ \text{otherwise} \end{cases}$
$$\sigma^r(w) = \begin{cases} \{u : \ w = ua\}, \\ \{w\}, \ \text{otherwise} \end{cases} \qquad \sigma^l(w) = \begin{cases} \{v : \ w = av\}, \\ \{w\}, \ \text{otherwise} \end{cases}$$
- If $\sigma \equiv \varepsilon \to a \in Ins_V$, then
$$\sigma^*(w) = \{uav : \ \exists u, v \in V^* \ (w = uv)\}, \ \sigma^r(w) = \{wa\}, \ \sigma^l(w) = \{aw\}.$$

$\alpha \in \{*, l, r\}$ expresses the way of applying an evolution rule to a word, namely at any position ($\alpha = *$), in the left ($\alpha = l$), or in the right ($\alpha = r$) end of the word, respectively. For every rule σ, action $\alpha \in \{*, l, r\}$, and $L \subseteq V^*$, we define the *α-action of σ on L* by $\sigma^\alpha(L) = \bigcup_{w \in L} \sigma^\alpha(w)$. Given a finite set of rules M, we define the *α-action of M* on the word w and the language L by:

$$M^\alpha(w) = \bigcup_{\sigma \in M} \sigma^\alpha(w) \text{ and } M^\alpha(L) = \bigcup_{w \in L} M^\alpha(w),$$

respectively. In what follows, we shall refer to the rewriting operations defined above as *evolutionary operations* since they may be viewed as linguistic formulations of local gene mutations. For two disjoint subsets P and F of an alphabet V and a word w over V, we define the predicates

$$\begin{aligned}
&\varphi^{(1)}(w; P, F) \equiv P \subseteq alph(w) && \wedge\ F \cap alph(w) = \emptyset \\
&\varphi^{(2)}(w; P, F) \equiv alph(w) \subseteq P && \\
&\varphi^{(3)}(w; P, F) \equiv P \subseteq alph(w) && \wedge\ F \not\subseteq alph(w) \\
&\varphi^{(4)}(w; P, F) \equiv alph(w) \cap P \neq \emptyset && \wedge\ F \cap alph(w) = \emptyset.
\end{aligned}$$

The construction of these predicates is based on *random-context conditions* defined by the two sets P (*permitting contexts*) and F (*forbidding contexts*).

For every language $L \subseteq V^*$ and $\beta \in \{(1), (2), (3), (4)\}$, we define:

$$\varphi^\beta(L, P, F) = \{w \in L \mid \varphi^\beta(w; P, F)\}.$$

An *evolutionary processor over* V is a tuple (M, PI, FI, PO, FO), where:

– Either ($M \subseteq Sub_V$) or ($M \subseteq Del_V$) or ($M \subseteq Ins_V$). The set M represents the set of evolutionary rules of the processor. As one can see, a processor is "specialized" in one evolutionary operation, only.
– $PI, FI \subseteq V$ are the *input* permitting/forbidding contexts of the processor, while $PO, FO \subseteq V$ are the *output* permitting/forbidding contexts of the processor (with $PI \cap FI = \emptyset$ and $PO \cap FO = \emptyset$).

We denote the set of evolutionary processors over V by EP_V.

An *accepting hybrid network of evolutionary processors* (AHNEP for short) is a 7-tuple $\Gamma = (V, U, G, N, \alpha, \beta, x_I, x_O)$, where:

– V and U are the input and network alphabet, respectively, $V \subseteq U$.
– $G = (X_G, E_G)$ is an undirected graph with the set of vertices X_G and the set of edges E_G. G is called the *underlying graph* of the network.
– $N : X_G \longrightarrow EP_U$ is a mapping which associates with each node $x \in X_G$ the evolutionary processor $N(x) = (M_x, PI_x, FI_x, PO_x, FO_x)$.
– $\alpha : X_G \longrightarrow \{*, l, r\}$; $\alpha(x)$ gives the action mode of the rules of node x on the words existing in that node.
– $\beta : X_G \longrightarrow \{(1), (2), (3), (4)\}$ defines the type of the *input/output filters* of a node. More precisely, for every node, $x \in X_G$, the following filters are defined:

$$\begin{aligned}
\text{input filter: } & \rho_x(\cdot) = \varphi^{\beta(x)}(\cdot; PI_x, FI_x), \\
\text{output filter: } & \tau_x(\cdot) = \varphi^{\beta(x)}(\cdot; PO_x, FO_x).
\end{aligned}$$

That is, $\rho_x(w)$ (resp. τ_x) indicates whether or not the string w can pass the input (resp. output) filter of x. More generally, $\rho_x(L)$ (resp. $\tau_x(L)$) is the set of strings of L that can pass the input (resp. output) filter of x.
– $x_I, x_O \in X_G$ are the *input* and the *output* node of Γ, respectively.

We say that $card(X_G)$ is the size of Γ. If α and β are constant functions, then the network is said to be *homogeneous*. In the theory of networks some types of underlying graphs are common, e.g., *rings, stars, grids*, etc. Networks of evolutionary processors with underlying graphs having these special forms have been considered in [1, 2, 11, 3]. We focus here on *complete* AHNEPs, i.e., AHNEPs having a complete underlying graph denoted by K_n, where n is the number of vertices.

A *configuration* of a AHNEP Γ as above is a mapping $C : X_G \longrightarrow 2^{V^*}$ which associates a set of strings with every node of the graph. A configuration may be understood as the sets of strings which are present in any node at a given moment. Given a string $w \in V^*$, the initial configuration of Γ on w is defined by $C_0^{(w)}(x_I) = w$ and $C_0^{(w)}(x) = \emptyset$ for all $x \in X_G - \{x_I\}$.

A configuration can change either by an *evolutionary step* or by a *communication step*. When changing by an evolutionary step, each component $C(x)$ of the configuration C is changed in accordance with the set of evolutionary rules M_x associated with the node x and the way of applying these rules $\alpha(x)$. Formally, we say that the configuration C' is obtained in *one evolutionary step* from the configuration C, written as $C \Longrightarrow C'$, iff $C'(x) = M_x^{\alpha(x)}(C(x))$ for all $x \in X_G$.

When changing by a communication step, each node processor $x \in X_G$ sends one copy of each string it has, which is able to pass the output filter of x, to all the node processors connected to x and receives all the strings sent by any node processor connected with x providing that they can pass its input filter.

Formally, we say that the configuration C' is obtained in *one communication step* from configuration C, written as $C \vdash C'$, iff $C'(x) = (C(x) - \tau_x(C(x))) \cup \bigcup_{\{x,y\}\in E_G}(\tau_y(C(y)) \cap \rho_x(C(y)))$ for all $x \in X_G$.

Let Γ be an AHNEP, the computation of Γ on the input string $w \in V^*$ is a sequence of configurations $C_0^{(w)}, C_1^{(w)}, C_2^{(w)}, \ldots$, where $C_0^{(w)}$ is the initial configuration of Γ on w, $C_{2i}^{(w)} \Longrightarrow C_{2i+1}^{(w)}$ and $C_{2i+1}^{(w)} \vdash C_{2i+2}^{(w)}$, for all $i \geq 0$. By the previous definitions, each configuration $C_i^{(w)}$ is uniquely determined by the configuration $C_{i-1}^{(w)}$. Otherwise stated, each computation in an AHNEP is deterministic. A computation *halts* (and it is said to be *finite*) if one of the following two conditions holds:

(i) There exists a configuration in which the set of strings existing in the output node x_O is non-empty. In this case, the computation is said to be an *accepting computation.*

(ii) There exist two consecutive identical configurations.

The *language accepted* by Γ is

$$L(\Gamma) = \{w \in V^* \mid \text{ the computation of } \Gamma \text{ on } w \text{ is an accepting one}\}.$$

3 Complexity Classes

The reader is referred to [8, 9] for the classical time and space complexity classes defined on the standard computing model of Turing machine.

We define some computational complexity measures by using AHNEP as the computing model. To this aim we consider a AHNEP Γ and the language L accepted by Γ. The *time complexity* of the accepting computation $C_0^{(x)}$, $C_1^{(x)}$, $C_2^{(x)}$, $\dots C_m^{(x)}$ of Γ on $x \in L$ is denoted by $Time_\Gamma(x)$ and equals m. The time complexity of Γ is the partial function from $\mathbf{N}$ to $\mathbf{N}$,

$$Time_\Gamma(n) = \max\{Time_\Gamma(x) \mid x \in L(\Gamma), |x| = n\}.$$

For a function $f : \mathbf{N} \longrightarrow \mathbf{N}$ we define

$$\mathbf{Time}_{AHNEP}(f(n)) = \{L \mid \text{ there exists an AHNEP } \Gamma \text{ and } n_0 \text{ such that } L = L(\Gamma) \text{ and } \forall n \geq n_0 (Time_\Gamma(n) \leq f(n))\}$$

Moreover, we write $\mathbf{PTime}_{AHNEP} = \bigcup_{k \geq 0} \mathbf{Time}_{AHNEP}(n^k)$.

Now we prove a result which establishes a strong connection between the complexity classes defined on Turing machines and those defined on AHNEPs.

Proposition 1. *For any nondeterministic Turing machine, M, recognizing a language L there exists an AHNEP, Γ, accepting the same language L. Moreover, if M works within time $f(n)$ then $Time_\Gamma(n) \in O(f(n))$.*

Proof. Let $M = (Q, V_1, V_2, \delta, q_0, B, F)$ be an arbitrary Turing machine. We define the new alphabets:

$U_1^{(K)} = \{\langle s, b, K, a\rangle \mid (s, b, K) \in \delta(q, a), s, q \in Q, a, b \in V_2 \setminus \{B\}\}, K \in \{R, L\}$,
$U_2 = \{[a, b] \mid a, b \in V_2 \setminus \{B\}\}$ $U_3 = \{X_a \mid a \in V_2 \setminus \{B\}\}$,
$U_4 = \{Y_a^{(b)} \mid a, b \in V_2 \setminus \{B\}\}$ $U_5 = \{Z_a \mid a \in V_2 \setminus \{B\}\}$,
$U_6 = \{W_a \mid a \in V_2 \setminus \{B\}\}$ $U_7 = \{\overline{sa} \mid s \in Q, a \in V_2 \setminus \{B\}\}$,
$U_8 = \{Y_a \mid a \in V_2 \setminus \{B\}\}$ $U_{10} = \{\tilde{sa} \mid s \in Q, a \in V_2 \setminus \{B\}\}$
$U_9^{(K)} = \{\langle\langle s, a, K, q\rangle\rangle \mid (s, a, K) \in \delta(q, B), s, q \in Q, a \in V_2 \setminus \{B\}\}, K \in \{R, L\}$.

Furthermore, for an alphabet T we denote by T' the alphabet consisting of the primed copies of all symbols in T. Now, we put

$$U = U_1^{(R)} \cup U_1^{(L)} \cup U_2 \cup U_3 \cup U_4 \cup U_5 \cup U_6 \cup U_7 \cup U_8 \cup U_9^{(R)} \cup U_9^{(L)} \cup U_{10} \cup V_2 \cup Q \cup U_3' \cup U_5' \cup U_6' \cup U_8' \cup (V_2 \setminus \{B\})'$$

We define the AHNEP $\Gamma = (V_1, U, G, N, \alpha, \beta, x_I, x_O)$, where G is a complete graph whose nodes are described below.

$M_{x_I} = \{\varepsilon \to q_0\}$, $PI_{x_I} = \emptyset$, $FI_{x_I} = U$, $PO_{x_I} = \emptyset$, $FO_{x_I} = \emptyset$, $\alpha(x_I) = r$, $\beta(x_I) = (1)$.

Table 1

Node	M	PI	FI	PO	FO	α	β
$x_1^{(\neq B)}$	$\{q \to \langle s, b, K, a\rangle \mid (s, b, K) \in \delta(q, a)\}$	$\emptyset$	$U \setminus (V_2 \cup Q)$	$\emptyset$	$\emptyset$	$*$	(1)
$x_1^{(\neq B)}(a, b)$	$\{\varepsilon \to Y_b^{(a)}\}$	$\{\langle s, b, R, a\rangle\}$	$U \setminus (V_2 \cup U_1^{(R)} \cup U_4)$	$\emptyset$	$\emptyset$	r	(1)
$x_1^{(\neq B)}(a)$	$\{a \to X_a\}$	$\{Y_b^{(a)}\}$	$U \setminus (V_2 \cup U_1^{(R)} \cup U_3 \cup U_4)$	U_3	$\emptyset$	$*$	(4)
$x_2^{(\neq B)}$	$\{\langle s, b, R, a\rangle \to s, Y_b^{(a)} \to b\}$	U_3	$U \setminus (V_2 \cup U_1^{(R)} \cup U_3 \cup U_4)$	$U \setminus (U_1^{(R)} \cup U_4)$	$U_1^{(R)} \cup U_4$	$*$	(4)
$x_3^{(\neq B)}$	$\{X_a \to \varepsilon\}$	U_3	$U \setminus (V_2 \cup U_3)$	$U \setminus U_3$	U_3	l	(4)

The nodes described in Table 1 are used for simulating a move of M which consists in reading a symbol different from B, possibly changing the state as well as the read symbol, and moving the tape head to the right. In this table, $s, q \in Q$, $a, b \in V_2 \setminus \{B\}$ and $K \in \{R, L\}$. Each table is accompanied by some explanations which emphasize the simulation mode. By the definition of the input node x_I, for any input string $w \in V_1^*$, $C_1^{(w)}(x_I) = \{wq_0\}$. In the next communication step both nodes $x_1^{(\neq B)}$ and $x_1^{(=B)}$ (which will be defined later) receive a copy of wq_0. Note that the initial ID of a computation of M on w is q_0w. Let us consider now an ID $\alpha q\beta$, which can be obtained by a computation in M starting with q_0w. By induction, we may assume that $\beta q\alpha \in C_m^{(w)}(x_1^{(\neq B)}) \cap C_m^{(w)}(x_1^{(=B)})$ for some $m \geq 1$. Let us suppose that $\beta = a\beta', a \in V_2 \setminus \{B\}, \beta' \in (V_2 \setminus \{B\})^*$. Clearly, $C_{m+1}^{(w)}(x_1^{(=B)}) \supseteq \{\beta\langle s, b, K, c\rangle\alpha \mid (s, b, K) \in \delta(q, c), s \in Q, b, c \in V_2 \setminus \{B\}, K \in \{R, L\}\}$. Obviously, only those strings with $c = a$ from the above ones are useful for our simulating process. Now, let us follow what happens with a string $\beta\langle s, b, R, a\rangle\alpha$ for some fixed $s \in Q, b \in V_2 \setminus \{B\}$ in the following steps. This string is accepted by $x_1^{(\neq B)}(a, b)$ only, where $Y_b^{(a)}$ is appended to its right-hand end. The resulting string $\beta\langle s, b, R, a\rangle\alpha Y_b^{(a)}$ is sent out by $x_1^{(\neq B)}(a, b)$ and $x_1^{(\neq B)}(a)$ is the unique node which can receive it. Here, exactly one occurrence of a in different copies of $\beta\langle s, b, R, a\rangle\alpha Y_b^{(a)}$ is replaced by X_a and all the obtained strings leave $x_1^{(\neq B)}(a)$. (We shall see later that only those strings starting with a in which this first occurrence of a is replaced by X_a can further navigate through the network; the others remain in $x_3^{(\neq B)}$ forever.) Then, all of them enter the node $x_2^{(\neq B)}$ where $\langle s, b, R, a\rangle$ and $Y_b^{(a)}$ are replaced by s and b, respectively. Both symbols must be replaced in two consecutive evolutionary steps since the output filter of $x_2^{(\neq B)}$ prevents leaving of this node by the strings containing symbols from $U_1^{(R)}$ or U_4. All the strings leaving $x_2^{(\neq B)}$ arrive in $x_3^{(\neq B)}$ where those starting with X_a can leave $x_3^{(\neq B)}$ after having removed X_a from their left-hand end, while the others remain in $x_3^{(\neq B)}$ forever. In this way, we check whether or not the first letter of β is indeed a. By the above explanations, we infer that

$$C_{m+14}^{(w)}(x_1^{(\neq B)}) \supseteq \{\beta' s\alpha b \mid (s, b, R) \in \delta(q, a), s \in Q, b \in V_2 \setminus \{B\}\}.$$

The nodes described in Table 2, together with $x_1^{(\neq B)}$ are used for simulating a move of M which consists in reading a symbol different from B, possibly changing the state as well as the read symbol, and moving the tape head to the left, provided that this is possible. In this table, $s, q \in Q$ and $a, b \in V_2 \setminus \{B\}$.

We start our explanation by returning to the configuration $C_{m+1}^{(w)}(x_1^{(=B)}) \supseteq \{\beta\langle s, b, K, c\rangle\alpha \mid (s, b, K) \in \delta(q, c), s \in Q, b, c \in V_2 \setminus \{B\}, K \in \{R, L\}\}$. In the sequel, we follow a string $\beta\langle s, b, L, a\rangle\alpha$ for some fixed $s \in Q, b \in V_2 \setminus \{B\}$. This string enters $x_2^{(\neq B)}(a, b)$ where, similarly to the situation described above when the followed string reached $x_1^{(\neq B)}(a)$, exactly one occurrence of a in different copies of $\beta\langle s, b, L, a\rangle\alpha$ is replaced by $[a, b]$. As we shall see later, the node $x_5^{(\neq B)}$ blocks all the strings obtained in $x_2^{(\neq B)}(a, b)$ which do not start with $[a, b]$ for further navigation through the network. Until that moment, we continue our explanations. The strings obtained in $x_2^{(\neq B)}(a,b)$ enter $x_2^{(\neq B)}(b)$, where W_b is appended to their right-hand end. Now, all these strings enter $x_4^{(\neq B)}$, where exactly one occurrence of each letter $c \in V_2 \setminus \{B\}$ is replaced by Z_c. The role of this node is to check whether or not $\alpha = \varepsilon$ since a move of the tape head to the left in the ID $\alpha q \beta$ is possible provided that $\alpha \neq \varepsilon$. More clearly, $C_{m+5}^{(w)}(x_4^{(\neq B)})$ has just received all strings of the form $\beta_1\langle s,b,R,a\rangle\alpha W_b$ and $\beta\langle s,b,R,a\rangle\alpha_1 W_b$, where β_1 and α_1 differ from β and α, respectively, on exactly one position where a in β or α is replaced by $[a,b]$.

Table 2

Node	M	PI	FI	PO	FO	α	β
$x_2^{(\neq B)}(a,b)$	$\{a \to [a,b]\}$	$\{\langle s,b,L,a\rangle\}$	$U \setminus (V_2 \cup U_1^{(L)})$	U_2	$\emptyset$	$*$	(4)
$x_2^{(\neq B)}(b)$	$\{\varepsilon \to W_b\}$	$\{[a,b]\}$	$U \setminus (V_2 \cup U_1^{(L)} \cup U_2)$	$\emptyset$	$\emptyset$	r	(1)
$x_4^{(\neq B)}$	$\{a \to Z_a\}$	U_6	$U \setminus (V_2 \cup U_1^{(L)} \cup U_2 \cup U_6)$	U_5	$\emptyset$	$*$	(4)
$x_5^{(\neq B)}$	$\{[a,b] \to \varepsilon\}$	U_5	$U \setminus (V_2 \cup U_1^{(L)} \cup U_2 \cup U_5 \cup U_6)$	$U \setminus U_2$	U_2	l	(4)
$x_3^{(\neq B)}(a)$	$\{\varepsilon \to W'_a\}$	$\{W_a\}$	$U \setminus (V_2 \cup U_1^{(L)} \cup U_5 \cup U_6)$	$U \setminus U_3$	U_3	l	(4)
$x_6^{(\neq B)}$	$\{W_a \to \varepsilon\}$	U'_6	$U \setminus (V_2 \cup U_1^{(L)} \cup U_5 \cup U_6 \cup U'_6)$	$U \setminus U_6$	U_6	r	(4)
$x_4^{(\neq B)}(a)$	$\{\varepsilon \to Z'_a\}$	$\{Z_a\}$	$U \setminus (V_2 \cup U_1^{(L)})$	$U \setminus U_3$	U_3	l	(4)
$x_7^{(\neq B)}$	$\{Z_a \to \varepsilon\}$	U'_5	$U \setminus (V_2 \cup U_1^{(L)} \cup U_5 \cup U'_5 \cup U'_6)$	$U \setminus U_5$	U_5	r	(4)
$x_8^{(\neq B)}$	$\{W'_a \to a\} \cup \{Z'_a \to a\} \cup \{\langle s,b,L,a\rangle \to s\}$	U'_5	$U \setminus (V_2 \cup U_1^{(L)} \cup U'_5 \cup U'_6)$	$U \setminus (U_1^{(L)} \cup U'_5 \cup U'_6)$	$U_1^{(L)} \cup U'_5 \cup U'_6$	$*$	(4)

Now, $C_{m+6}^{(w)}(x_4^{(\neq B)})$ contains all strings $h(\beta_1)\langle s,b,L,a\rangle\alpha W_b$, $\beta_1\langle s,b,L,a\rangle h(\alpha) W_b$, $\beta\langle s,b,L,a\rangle h(\alpha_1) W_b$, $h(\beta)\langle s,b,L,a\rangle\alpha_1 W_b$, where $h : ((V_2 \setminus \{B\}) \cup U_2 \cup U_6 \cup U_1^{(L)})^* \longrightarrow 2^{((V_2\setminus\{B\})\cup U_2 \cup U_6 \cup U_1^{(L)} \cup U_5)^*}$ is a finite substitution which leaves unchanged all the symbols from $U_2 \cup U_6 \cup U_1^{(L)}$ and $h(c) = \{c, Z_c\}$, for all $c \in V_2 \setminus \{B\}$. But $C_{m+6}^{(w)}(x_4^{(\neq B)})$ contains no string $\beta_1\langle s,b,L,a\rangle\alpha W_b$ or $\beta\langle s,b,L,a\rangle\alpha_1 W_b$ from above. Later, it will turn out that only the strings $[a,b]\beta'\langle s,b,L,a\rangle\alpha' Z_c W_b$ are useful for the rest of computation. Indeed, the strings which do not start with a symbol in U_2 remain blocked in $x_5^{(\neq B)}$. The others leave $x_5^{(\neq B)}$ and enter $x_3^{(\neq B)}(a)$ where they receive W'_a in their left-hand end, provided that they have W_a in their right-hand end. After that, W_a is deleted. This is actually the way of rotating a symbol from the right-hand end to the left-hand end of a string. The role of $x_4^{(\neq B)}(a)$ and $x_7^{(\neq B)}$ is the same and now we can easily notice that only the strings proceeding from $[a,b]\beta'\langle s,b,L,a\rangle\alpha' Z_c W_b$ can continue the computation. Finally, we deduce that

$C_{m+22}^{(w)}(x_1^{(\neq B)}) \supseteq \{cb\beta' s\alpha' \mid \alpha = c\alpha', (s,b,L) \in \delta(q,a), s \in Q, b,c \in V_2 \setminus \{B\}\}$.
The nodes described in Table 3 are used for simulating a move of M which consists in reading B and changing it into a symbol from $V_2 \setminus \{B\}$, possibly changing the current state, and moving the tape head to the right. In this table, $s,q \in Q$, $a \in V_2 \setminus \{B\}$ and $K \in \{R,L\}$.

Table 3

Node	M	PI	FI	PO	FO	α	β
$x_1^{(=B)}$	$\{\varepsilon \to \langle\langle s,a,K,q\rangle\rangle \mid (s,a,K) \in \delta(q,B)\}$	$\emptyset$	$U \setminus (V_2 \cup Q)$	$\emptyset$	$\emptyset$	r	(1)
$x_1^{(=B)}(q)$	$\{q \to \varepsilon\}$	$\{\langle\langle s,a,K,q\rangle\rangle\}$	$U \setminus (V_2 \cup U_9 \cup Q)$	U	$\emptyset$	l	(4)
$x_1^{(=B)}(s,a)$	$\{\varepsilon \to \overline{sa}\}$	$\{\langle\langle s,a,R,q\rangle\rangle\}$	$U \setminus (V_2 \cup U_9^{(R)})$	$\emptyset$	$\emptyset$	l	(1)
$x_1^{(=B)}(a)$	$\{\varepsilon \to Y_a\}$	$\{\overline{sa}\}$	$U \setminus (V_2 \cup U_9^{(R)} \cup U_7)$	U	$\emptyset$	r	(4)
$x_2^{(=B)}$	$\{\langle\langle s,a,R,q\rangle\rangle \to \varepsilon\}$	U_8	$U \setminus (V_2 \cup U_9^{(R)} \cup U_7 \cup U_8)$	$\emptyset$	$U_9^{(R)}$	$*$	(1)
$x_3^{(=B)}$	$\{Y_a \to a\} \cup \{\overline{sa} \to s\}$	U_8	$U \setminus (V_2 \cup U_8 \cup U_7)$	$U \setminus (U_8 \cup U_7)$	$U_8 \cup U_7$	$*$	(4)

We consider a string $\beta q\alpha \in C_m^{(w)}(x_1^{(=B)})$ and $(s,a,R) \in \delta(q,B)$ a transition which the move of M we want to simulate is based on. First, $\beta q\alpha\langle\langle s,a,R,q\rangle\rangle$ is produced in $x_1^{(=B)}$ and then sent out. The string enters $x_1^{(=B)}(q)$ where one checks whether or not $\beta = \varepsilon$. Only $q\alpha\langle\langle s,a,R,q\rangle\rangle$, after deleting q, is able to leave $x_1^{(=B)}(q)$, the others being blocked in this node. Now, $\alpha\langle\langle s,a,R,q\rangle\rangle$ enters $x_1^{(=B)}(s,a)$, where the symbol $\overline{sa}$ is appended to its left-hand end, and the resulting string enters $x_1^{(=B)}(a)$, where Y_a is appended to its right-hand end.

Table 4

Node	M	PI	FI	PO	FO	α	β
$x_2^{(=B)}(s,a)$	$\{\varepsilon \to \tilde{sa}\}$	$\{\langle\langle s,a,L,q\rangle\rangle\}$	$U \setminus (V_2 \cup U_9^{(L)})$	U	$\emptyset$	l	(4)
$x_4^{(=B)}$	$\{\langle\langle s,a,L,q\rangle\rangle \to \varepsilon\}$	U_7	$U \setminus (V_2 \cup U_9^{(L)} \cup U_{10})$	$U \setminus U_9^{(L)}$	$U_9^{(L)}$	$*$	(4)
$x_2^{(=B)}(a)$	$\{\varepsilon \to X'_a\}$	$\{\tilde{sa}\}$	$U \setminus (V_2 \cup U_{10})$	U	$\emptyset$	l	(4)
$x_5^{(=B)}$	$\{a \to Y'_a\}$	U'_3	$U \setminus (V_2 \cup U_{10} \cup U'_3)$	U'_8	$\emptyset$	$*$	(4)
$x_3^{(=B)}(a)$	$\{\varepsilon \to a'\}$	$\{Y'_a\}$	$U \setminus (V_2 \cup U_{10} \cup U'_3 \cup U'_8)$	$\emptyset$	$\emptyset$	l	(1)
$x_6^{(=B)}$	$\{Y'_a \to \varepsilon\}$	V'_2	$U \setminus (V_2 \cup U_{10} \cup U'_3 \cup U'_8 \cup V'_2)$	$U \setminus U'_8$	U'_8	r	(4)
$x_7^{(=B)}$	$\{X'_a \to a\} \cup \{\tilde{sa} \to s\} \cup \{a' \to a\}$	U'_3	$U \setminus (V_2 \cup U_{10} \cup U'_3 \cup V'_2)$	$U \setminus (V'_2 \cup U'_3 \cup U_{10})$	$V'_2 \cup U'_3 \cup U_{10}$	$*$	(4)

Then, $\langle\langle s,a,R,q\rangle\rangle$is removed and Y_a, as well as $\overline{sa}$, are replaced by a and s, respectively. Hence

$$C^{(w)}_{m+14}(x^{(=B)}_1) \supseteq \{s\alpha a \mid (s,a,R) \in \delta(q,B), s \in Q, a \in V_2 \setminus \{B\}\}.$$

The nodes described in Table 4 are used, together with the nodes $x^{(=B)}_1$ and $x^{(=B)}_1(q)$, $q \in Q$, for simulating a move of M which consists in reading B and changing it into a symbol from $V_2 \setminus \{B\}$, possibly changing the current state, and moving the tape head to the left. In this table, $s,q \in Q$ and $a \in V_2 \setminus \{B\}$.

We consider again a string $\beta q\alpha \in C^{(w)}_m(x^{(=B)}_1)$ and $(s,a,L) \in \delta(q,B)$ a transition which the move of M we want to simulate is based on. As above, after producing $\beta q\alpha\langle\langle s,a,L,q\rangle\rangle$ in $x^{(=B)}_1$, this string enters $x^{(=B)}_1(q)$, where one checks whether or not $\beta = \varepsilon$ and q is removed. Then, $\alpha\langle\langle s,a,L,q\rangle\rangle$ enters $x^{(=B)}_2(s,a)$, where $\tilde{sa}$ is appended to its left-hand end. The new string, after having removed $\langle\langle s,a,L,q\rangle\rangle$ receives X'_a in its left-hand end resulting in $X'_a\tilde{sa}\alpha$. Now, the last symbol of α, say b, is shifted as b' before X'_a by means of the nodes $x^{(=B)}_5$, $x^{(=B)}_6$, and $x^{(=B)}_3(b)$. The obtained string is now $b'X'_a\tilde{sa}\alpha'$, with $\alpha = \alpha'b$. Therefore,

$$C^{(w)}_{m+22}(x^{(=B)}_1) \supseteq \{basa' \mid (s,a,L) \in \delta(q,B), s \in Q, a \in V_2 \setminus \{B\}, \alpha = \alpha'b\}.$$

The construction of Γ is completed with the output node x_O defined by $M_{x_O} = \emptyset$, $PI_{x_O} = F$, $FI_{x_O} = U\setminus(V_2 \cup F)$, $PO_{x_O} = \emptyset$, $FO_{x_O} = U$, $\alpha(x_O) = *$, $\beta(x_O) = (4)$. By the aforementioned explanations, we infer that $L(M) = L(\Gamma)$.

It is worth mentioning that the underlying graph G is the complete graph K_p, with $p = 15+7(\text{card}(V_2)-1)+\text{card}(Q)+2(\text{card}(V_2)-1)^2+2\text{card}(Q)(\text{card}(V_2)-1)$. That is, the number of nodes of Γ is bounded by a quadratic function depending on the number of states and symbols of M. Also, the total number of symbols used by Γ is the above simulation is bounded by a cubic function depending on the number of states and symbols of M. More precisely,

$$\begin{aligned}\text{card}(U) = 4\text{card}(Q)(\text{card}(V_2)-1)^2 + 2(\text{card}(V_2)-1)^2 + \text{Card}(V_2) + \\ 2\text{card}(Q)(\text{card}(V_2)-1) + 9(\text{card}(V_2)-1) + \text{card}(Q)\end{aligned} \qquad \square$$

Now we are ready to prove the main result of this paper.

Theorem 1. NP = PTime$_{AHNEP}$.

Proof. Let L be a language accepted by a nondeterministic Turing machine M with k tapes such that for each $x \in L$, $|x| = n$, M can accept x in no more than $p(n)$ moves. We write this as $T_M(n) \le p(n)$. Clearly, we can construct a Turing machine M' such that $T_{M'}(n) \le p(n)/\sqrt{22}$. By the well-known results regarding tape compression, we can construct a Turing machine M'' with one tape only, such that $T_{M''}(n) \le p^2(n)/22$. Now, by the previous proof, we construct an AHNEP Γ such that $L(M'') = L(\Gamma)$ and $Time_\Gamma(n) \le 22T_{M''} \le p^2(n)$, which concludes the proof of $\mathbf{NP} \subseteq \mathbf{PTime}_{AHNEP}$.

Conversely, let L be a language accepted by an AHNEP Γ in polynomial time $p(n)$. We construct a nondeterministic Turing machine M as follows:

(1) M has a finite set of states associated with each node of Γ. This set is divided into disjoint subsets such that each filter (input or output) and each rule has an associated subset of states.

(2) M chooses nondeterministically a copy of the input word from those existing in the initial node of Γ (this word is actually on the tape of M in its initial ID) and follows its itinerary through the underlying network of Γ. Let us suppose that the contents of the tape of M is α; M works according to the following strategy labelled by $(*)$:

(i) When M enters a state from the subset of states associated to a rule $a \to b$, it applies this rule to an occurrence of a in α, if any, nondeterministically chosen. If α does not contain any occurrence of a, M blocks the computation.

(ii) When M enters a state from the subset of states associated to a filter, it checks whether α can pass that filter. If α does not pass it, M blocks the computation. Clearly, M checks first the condition of the current node (sending node) output filter and then the condition of the receiving node input filter (which becomes the current node).

(iii) As soon as M has checked the input filter condition of the output node of Γ, it accepts its input word.

It is rather plain that M accepts L. If the input word w in the initial node of Γ is in L, then there exists a computation in Γ of time complexity $O(p(|w|))$. Since in any evolutionary step one inserts at most one letter, the length of α in $(*)$ is at most $p(|w|) + |w|$. Clearly, each step (i) and (ii) of $(*)$ can be accomplished in time $O(|\alpha|)$. Therefore, w is accepted by M in $O(p^2(|w|))$ time and we are done. $\square$

References

1. J. Castellanos, C. Martin-Vide, V. Mitrana, J. Sempere, Solving NP-complete problems with networks of evolutionary processors, *IWANN 2001* (J. Mira, A. Prieto, eds.), LNCS 2084, Springer-Verlag, 2001, 621–628.
2. J. Castellanos, C. Martin-Vide, V. Mitrana, J. Sempere, Networks of evolutionary processors, *Acta Informatica* 39 (2003), 517-529.
3. J. Castellanos, P. Leupold, V. Mitrana, Descriptional and computational complexity aspects of hybrid networks of evolutionary processors, submitted.
4. E. Csuhaj-Varjú, A. Salomaa, Networks of parallel language processors. In: *New Trends in Formal Languages* (Gh. Păun, A. Salomaa, eds.), LNCS 1218, Springer Verlag, 1997, 299–318.
5. E. Csuhaj-Varjú, V. Mitrana, Evolutionary systems: a language generating device inspired by evolving communities of cells, *Acta Informatica* 36 (2000), 913–926.
6. L. Errico, C. Jesshope, Towards a new architecture for symbolic processing. In *Artificial Intelligence and Information-Control Systems of Robots '94* (I. Plander, ed.), World Sci. Publ., Singapore, 1994, 31–40.
7. S. E. Fahlman, G. E. Hinton, T. J. Seijnowski, Massively parallel architectures for AI: NETL, THISTLE and Boltzmann machines. In *Proc. AAAI National Conf. on AI*, William Kaufman, Los Altos, 1983, 109–113.
8. J. Hartmanis, P.M. Lewis II, R.E. Stearns, Hierarchies of memory limited computations. *Proc. 6th Annual IEEE Symp. on Switching Circuit Theory and Logical Design*, 1965, 179 - 190.
9. J. Hartmanis, R.E. Stearns, On the computational complexity of algorithms, *Trans. Amer. Math. Soc.* 117 (1965), 533–546.

10. W. D. Hillis, *The Connection Machine*, MIT Press, Cambridge, 1985.
11. C. Martin-Vide, V. Mitrana, M. Perez-Jimenez, F. Sancho-Caparrini, Hybrid networks of evolutionary processors, *Proc. of GECCO 2003*, LNCS 2723, Springer Verlag, Berlin, 401 - 412.
12. D. Sankoff et al. Gene order comparisons for phylogenetic inference:Evolution of the mitochondrial genome. *Proc. Natl. Acad. Sci. USA*, 89 (1992) 6575–6579.

Building the Components for a Biomolecular Computer

Clint Morgan[1], Darko Stefanovic[1], Cristopher Moore[1],
and Milan N. Stojanovic[2]

[1] Department of Computer Science, University of New Mexico
{clint, darko, moore}@cs.unm.edu
[2] Department of Medicine, Columbia University
mns18@columbia.edu

Abstract. We propose a new method for amorphous bio-compatible computing using deoxyribozyme logic gates [1] in which oligonucleotides act as enzymes on other oligonucleotides, yielding oligonucleotide products. Moreover, these reactions can be controlled by inputs that are also oligonucleotides. We interpret these reactions as logic gates, and the concentrations of chemical species as signals. Since these reactions are *homogeneous*, i.e., they use oligonucleotides as both inputs and outputs, we can compose them to construct complex logic circuits. Thus, our system for chemical computation offers functionality similar to conventional electronic circuits with the potential for deployment inside of living cells. Previously, this technology was demonstrated in closed-system batch reactions, which limited its computational ability to simple feed-forward circuits. In this work, we go beyond closed systems, and show how to use thermodynamically open reactors to build biomolecular circuits with feedback. The behavior of an open chemical system is determined both by its chemical reaction network and by the influx and efflux of chemical species. This motivates a change in design process from that used with closed systems. Rather than focusing solely on the stoichiometry of the chemical reactions, we must carefully examine their kinetics. Systems of differential equations and the theory of dynamical systems become the appropriate tools for designing and analyzing such systems. Using these tools, we present an *inverter*. Next, by introducing feedback into the reaction network, we construct devices with a sense of state. We show how a combination of analytical approximation techniques and numerical methods allows us to tune the dynamics of these systems. We demonstrate a flip-flop which exhibits behavior similar to the RS flip-flop of electronic computation. It has two states in which the concentration of one oligonucleotide is high and the other is low or vice versa. We describe how to control the state of the flip-flop by varying the concentration of the substrates. Moreover, there are large regions of parameter space in which this behavior is robust, and we show how to tune the influx rates as a function of the chemical reaction rates in a way that ensures bistability.

1 Introduction

We use deoxyribozymes (nucleic acid enzymes) as gates to transform input and substrate signals (molecular concentrations) into product signals and thereby perform simple computation. Since the inputs are of the same type as the outputs, viz. oligonucleotides, gates may, in principle, be connected in complex circuits, with the output of one gate acting as the input of another. Thus, we may design chemical systems that perform complex computations from simple boolean primitives in much the same way electronic computers are built from simple logic gates. These devices could operate without macroscopic intervention in a biological environment, and the goal of this technology is autonomous *in vivo* computation for diagnostic and therapeutic purposes. We have reported gates with a single layer of logic, and no inter-gate communication [1]. Devices that function as a half-adder [2] and a tic-tac-toe automaton [3] have been built and tested in the laboratory.

These gates have been deployed in a closed reactor, which effectively limits this technology to one-shot boolean computations. To overcome this limitation, we explore using this chemistry in an open reactor, in which gates could be re-used many times and connected in recurrent, rather than feed-forward, circuits. This adds a level of complexity to the engineering task, but we develop a process that may be used to engineer these devices. We apply methods of dynamical systems to construct reaction networks in open reactors that implement rudimentary elements of digital chemical computation. This allows us to investigate complex reaction networks that make use of inter-gate communication and feedback.

2 The Chemical Kinetics of Deoxyribozyme Logic Gates

The four components of our deoxyribozyme system are inputs, gates, substrates, and products. Under certain input conditions a gate is an active enzyme [1]. The effect of input molecules on the catalytic activity of the gate defines the logic operation that the gate performs. A gate requires the presence and/or absence of certain inputs to be active. When active, the enzymatic gate is a phosphodiesterase: it catalyzes an oligonucleotide cleavage reaction. A substrate molecule is cleaved into two product molecules. The product molecules represent the output signal of the gate. Computations are carried out in solution, where gates communicate by diffusion of oligonucleotides. Logic signals, true or false, are expressed by high or low concentrations of specific oligonucleotides. Oligonucleotides transmit information by participating in the reactions of multiple gates. The simplest example is an oligonucleotide that is a product of one gate and an input to another; serving as a substrate would suffice as well.

The mechanism of a deoxyribozyme gate is as follows. Input molecules bind to the designated locations on the gate molecules. The binding of an input to a gate affects the conformation of the gate, which in turn affects catalytic activity. Under appropriate circumstances, the gate is an active enzyme, in which case it binds to a substrate molecule, cleaves it into two molecules of product, and separates into two molecules of product and one active gate complex. Active gates continue to operate as long as there is substrate remaining to be cleaved.

In order to design larger circuits, we must first understand the dynamic behavior of individual logic gates. We set up the experiment as follows. We prepare a solution with a concentration of $G = 250\,\text{nM}$ of a specific YES gate (which becomes active in the presence of input), a certain concentration I of the matching input, and a

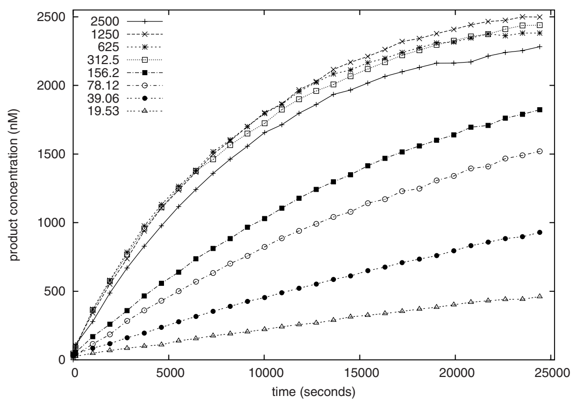


Fig. 1. Measured kinetics of a deoxyribozyme gate for different input concentrations I (nM)

concentration $S = 2500\,\text{nM}$ of the substrate cleaved by the gate. At 900s intervals we record the instrumentally measured fluorescence. We repeat the experiment varying I, starting with $I = S$ and repeatedly halving it. The measured fluorescence level of a molecular species is proportional to its concentration. The specific fluorescences of the product and the substrate have been established separately and are in a ratio of 8:1. Therefore the increase of total fluorescence is proportional to the amount of product, which allows us to convert measured fluorescence into product concentration, shown in Figure 1.

For small values of the input concentration I, product concentration P rises linearly with time, with slope proportional to I. For larger values of I, the growth soon reaches a plateau defined by the initial substrate concentration S: when all of the substrate has been converted to product, the reaction stops. Note also the saturating behavior: whereas the slope of $t \mapsto P$ increases with I for small I, it remains roughly constant for $I > G$.

We will model the kinetics of the deoxyribozyme gates as follows. First we note that the cleavage and separation of substrate molecules is the slowest of the reactions—it is the rate-limiting process. We will assume that bonding between gate and input molecules is instantaneous and complete. Thus, the number of active gates at a given time is a simple calculation depending solely on the number of gates and inputs.

Cleavage of substrate requires both substrate and an active gate complex. Experiments have shown that the rate of production is proportional to the concentration of both reactants. Hence, a model for the rate at which product is produced is: $\frac{dP}{dt} = \beta S G_A$, where P is the product concentration, β is the reaction rate constant, S is the substrate concentration, and G_A is the concentration of *active* gates. In the experiments shown in Figure 1, I (and thus G_A) was held constant. Therefore, the solution is an exponential decay of the substrate S. This model agrees well with observed measurements in Figure 1, and analysis of the measured data gives a rough estimate of the reaction rate constant $\beta = 5 \cdot 10^{-7}\,\text{nM}^{-1}\text{s}^{-1}$. This value will be used to model the chemical gates in the circuit designs presented herein.

3 The Reactor

Chemical reactors may be divided into closed systems, where reactants are added to a solution and the reaction is allowed to proceed toward equilibrium, and open systems, where reactants are continuously supplied and excess solution is removed. We explore the benefits and design considerations associated with using our chemistry in an open reactor. Previous theoretical work has described circuits created from hypothetical enzymatic transistors in open reactors [4]. We expand on this work by presenting circuit designs based on the chemical technology described above. Reactions in a closed environment are subject to the Second Law of Thermodynamics, which posits that the free energy in a closed system will continuously decrease and implies that the system will move toward an equilibrium. This does not rule out interesting behavior, such as oscillations on the way to an equilibrium [5], but it implies a finite number of cycles through these oscillations. Thus it would be impossible to implement a recurrent digital circuit in which gates could switch on and off an arbitrary number of times.

Instead, we use a thermodynamically open system; material is continuously supplied and removed, as in a living cell. The circuit may be reused and produce many outputs over its lifetime, so that it is recurrent rather than feed-forward. While a long-term goal of this technology is deployment inside of (thermodynamically open) living cells, the first step toward that end is testing and verification in a laboratory setting. A model open environment is the continuous-flow stirred tank reactor. It delivers reactants into a reaction chamber, stirred to maintain a uniform distribution of chemical species. An outflow removes solution from the reactor to maintain constant volume. The inputs can be varied in terms of their concentrations in the input solution and the flow rate into the reactor. Both the concentration and the *volumetric* influx of a solution can be varied while still maintaining the same total *molecular* influx rate. Thus we can manipulate total efflux while maintaining desired concentrations of chemical species inside the reactor.

The decay rate of the reactor (k) is equal to the efflux rate (E) divided by the volume (V). As the decay rate is increased, material spends less time inside the reactor. Because the reactor state changes faster, the circuit speeds up. However, this increases the amount of chemical species needed to maintain the same concentrations. In the specific design below, the reactor will have a total efflux of $5 \cdot 10^{-8}\,\mathrm{m^3 s^{-1}}$ and a volume of $5 \cdot 10^{-4}\,\mathrm{m^3}$. This results in a decay rate of $10^{-4}\,\mathrm{s^{-1}}$. While the resulting circuits will operate *very* slowly, these values were chosen as design points corresponding to equipment that can be found in a traditional chemistry laboratory.

4 A Simple Inverter

We begin by examining a simple computational device: the digital inverter. It is built from a single type of Not gate G operating in an open reactor. The reactor is supplied with a constant influx of gate and substrate molecules. In addition, input I is supplied to provide an external drive. The output of this inverter is expressed by the concentration of product P cleaved from substrate S. The behavior of the system can be modeled with the following system of four coupled differential equations:

$$\frac{dG}{dT} = \frac{G^m - EG(T)}{V} \tag{1}$$

$$\frac{dI}{dT} = \frac{I^m - EI(T)}{V} \tag{2}$$

$$\frac{dP}{dT} = \beta S(T)\max(0, G(T) - I(T)) - \frac{EP(T)}{V} \tag{3}$$

$$\frac{dS}{dt} = \frac{S^m}{V} - \beta S(T)\max(0, G(T) - I(T)) - \frac{ES(T)}{V} \tag{4}$$

where I^m, G^m, and S^m are the constant rates of *molar* influx of the respective chemical species, V is the volume of the reactor, E is the rate of volume efflux, and β is the reaction rate constant. The max terms in (3) and (4) come from our assumption that the binding of input to gate molecule is both instantaneous and complete.

Clearly, this system can function as an inverter. If there are no inputs in the reactor, all of the gates are active and produce product—this is the high signal. As input is added gates become inhibited, and the product concentration falls. As the input concentration reaches the gate concentration, all gates become inhibited and the product concentration falls to zero—this is the low signal.

To explore the equilibrium behavior of the inverter we first assume that the input concentration never exceeds the gate concentration; we can then eliminate the max functions from equations (1)-(4). We can now set the derivatives to zero and solve for P. This produces the following relation between input concentration and output (product) concentration:

$$P = \frac{\beta S^m V(\frac{G^m}{E} - I)}{E^2 + \beta V E(\frac{G^m}{E} - I)} = \frac{\frac{S^m}{E}(\frac{G^m}{E} - I)}{\frac{E}{\beta V} + \frac{G^m}{E} - I}$$

Introducing rescaling parameters $\alpha = \frac{S^m}{E}$, $\gamma = \frac{G^m}{E}$, and $\delta = \frac{E}{\beta V}$ allows further simplification. Thus we arrive at the following equation for the static transfer curve:

$$P = \frac{\alpha(\gamma - I)}{\delta + (\gamma - I)}$$

This shows how the output concentration P depends on the input concentration I.

Further constraints must be introduced to create an inverter with well-defined signal levels. First, the concentration corresponding to a high logic value is defined as H. We require that $P = H$ when $I = 0$ and $P = 0$ when $I = H$. These conditions yield the constraints $\gamma = H$ and $\alpha = H + \delta$. Thus, to alter the static transfer curve, we may vary the parameters δ and α, while maintaining the relationship: $\alpha = H + \delta$. Figure 2 shows the transfer functions obtained by setting δ to a range of values. As δ is increased, the curve flattens out and becomes close to linear. As δ is decreased the curve becomes more bowed out (i.e., a large derivative).

This transfer curve is far from the sigmoid shape desired in digital computing. Any noise that moves the input concentration away from its digital value will propagate through to the output, possibly resulting in computational errors. However, we may construct inverters with differing static characteristics by concatenating several gates in a cascade. Details of this construction are given in [6].

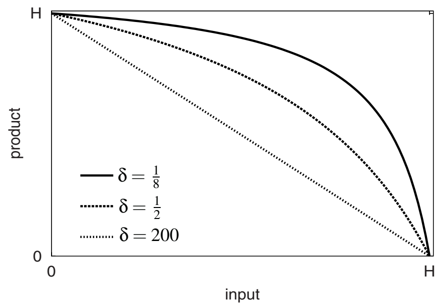


Fig. 2. The static transfer curve for an inverter constructed from a NOT gate in an open system with several different values for δ: $\frac{1}{8}$, $\frac{1}{2}$, and 200. As δ increases the transfer curve approaches a straight line from high product and low input to low product and high input concentrations

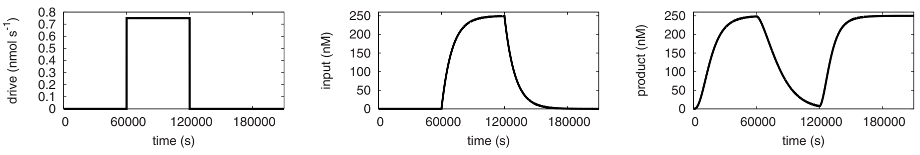


Fig. 3. Left: external drive (molecular influx), as input to the circuit. Middle: concentration of input I inside the reactor. Right: product concentration P, the output of the inverter. The input concentration is moved from low, to high, and back to low at $6 \cdot 10^4$ s intervals

While the static behavior addresses equilibrium characteristics, this analysis neglects the dynamic behavior of the system. Ultimately, the static transfer characteristic depends on only one ratio, but the dynamic behavior is less restricted and depends on several variables. However, this solution space is narrowed by the physical restrictions of our technology. We define a logical high value to be a concentration of 250 nM and calculate the rest of the parameters for the system. Figure 3 shows the results of numerical integration of the inverter working under such conditions. The reactor was started with zero concentration of all chemical species. The propagation delay of the inverter is $\approx 7.9 \cdot 10^3$ s, with a $t_{PHL} \approx 12.5 \cdot 10^3$ s, and $t_{PLH} \approx 3.2 \cdot 10^3$ s.

5 A Chemical Flip-Flop

Moving to an open system allows us to construct recurrent chemical and logical circuits—circuits with a lasting internal state or memory, which can change and be accessed over time. The simplest such system in digital logic is a *flip-flop*. This is simply a bistable system, which exhibits three behaviors depending on its inputs, commonly called *hold*, *set*, and *reset*. In the hold behavior, there are two stable states, which represent high and low outputs of the system. Set forces the system into its high stable state regardless of its previous state; similarly, reset forces it to its low stable state. Thus a flip-flop represents a single bit of memory, which can be stored (hold) or overwritten (set or reset).

A system that functions as a flip-flop can be constructed with a network of two NOT gates connected in a cycle of inhibition. A gate G_1 cleaves substrate S_1 to produce product P_1, which inhibits the catalytic activity of gate G_2; gate G_2 cleaves S_2 to produce P_2, which inhibits G_1 to complete the cycle. Output from the flip-flop is in terms of the concentration of the cleaved product P_2, with high or low concentration corresponding to a logical one or zero. The flip-flop is controlled by varying the influx of substrates 1 and 2 to the reactor, while gates 1 and 2 are continuously supplied.

We define constants G_1^m and G_2^m to be the rates of molecular influx of gate solutions. The external control is modeled by the functions $S_1^m(T)$ and $S_2^m(T)$, which describe the variable *molecular* influx of substrates at time T. The rate of efflux of the system is given by E. We define $P_1(T)$, $P_2(T)$, $S_1(T)$, $S_2(T)$, $G_1(T)$, and $G_2(T)$ to be the concentrations within the reactor at time T of product 1, product 2, substrate 1, substrate 2, gate 1, and gate 2, respectively. The system's dynamics are modeled by the following system of six coupled differential equations:

$$\begin{aligned}
\frac{dG_1}{dT} &= \frac{G_1^m - EG_1(T)}{V}\\
\frac{dG_2}{dT} &= \frac{G_2^m - EG_2(T)}{V}\\
\frac{dP_1}{dT} &= \beta_1 S_1(T)\max(0, G_1(T) - P_2(T)) - \frac{EP_1(T)}{V}\\
\frac{dP_2}{dT} &= \beta_2 S_2(T)\max(0, G_2(T) - P_1(T)) - \frac{EP_2(T)}{V}\\
\frac{dS_1}{dT} &= \frac{S_1^m(T)}{V} - \beta_1 S_1(T)\max(0, G_1(T) - P_2(T)) - \frac{ES_1(T)}{V}\\
\frac{dS_2}{dT} &= \frac{S_2^m(T)}{V} - \beta_2 S_2(T)\max(0, G_2(T) - P_1(T)) - \frac{ES_2(T)}{V}
\end{aligned}$$

where β_1 and β_2 are the reaction rate constants and V is the volume of the reactor.

We now examine the model to determine the conditions under which it will function as a flip-flop. Since we control it using substrate concentrations, we must determine how its output depends on these. To simplify the analysis, we take the gate concentrations to be constant (depending only on the efflux of the system). Hence we view the substrate concentrations as parameters of the system rather than dynamic variables. Now the behavior of the flip-flop can be understood using the following two-dimensional system:

$$\frac{dp_1}{dt} = r_1\max(0, g_1 - p_2(t)) - kp_1(t) \tag{5}$$

$$\frac{dp_2}{dt} = r_2\max(0, g_2 - p_1(t)) - kp_2(t) \tag{6}$$

where p_1 and p_2 are the product concentrations, r_1 and r_2 are lumped parameters representing substrate concentration and the reaction rate constant for the two reactions, g_1 and g_2 are the gate concentrations, and k is the decay constant. In order to function as a flip-flop, the system must have two stable states: a *set* state with a high value of p_2, and a *reset* state with a low value of p_2. Additionally there should exist control mechanisms to switch between states, in this case by varying the substrate concentrations. High concentration of both substrates is used to hold the flip-flop state, while the absence of one is used to set or reset the flip-flop. We now examine how the system will behave for various values of the parameters r_1 and r_2.

We begin our analysis by examining the *nullclines* of the system, i.e., the curves along which the time derivatives of the variables are constant. Setting equations (5) and (6) to zero yields:

$$\frac{dp_1}{dt} = 0 \Rightarrow p_1 = \frac{r_1 \max(0, g_1 - p_2)}{k}$$
$$\frac{dp_2}{dt} = 0 \Rightarrow p_2 = \frac{r_2 \max(0, g_2 - p_1)}{k}$$

The derivative on the nullclines will have components in a single direction, along either the p_1 or p_2 axis. This observation can be further qualified by inspecting equations (5)-(6). Notice that as we increase the value of p_1 the value of $\frac{dp_1}{dt}$ decreases, and as we decrease the value of p_1 the value of $\frac{dp_1}{dt}$ increases. This same relationship holds between p_2 and $\frac{dp_2}{dt}$. Thus a point above the p_1 nullcline will have a derivative in the negative p_1 direction and a point below the p_1 nullcline will have a derivative in the positive p_1 direction. We can use these facts to partition the phase space into regions where the signs of the derivatives are known [7].

Figure 4 shows two possible configurations of the nullclines along with the signs of the derivatives in different regions of the phase space. In the case shown in Figure 4**a** the two nullclines intersect (to create fixed points) at points A, B, and C. The stability of these points can be investigated by examining the slopes of the surrounding regions.

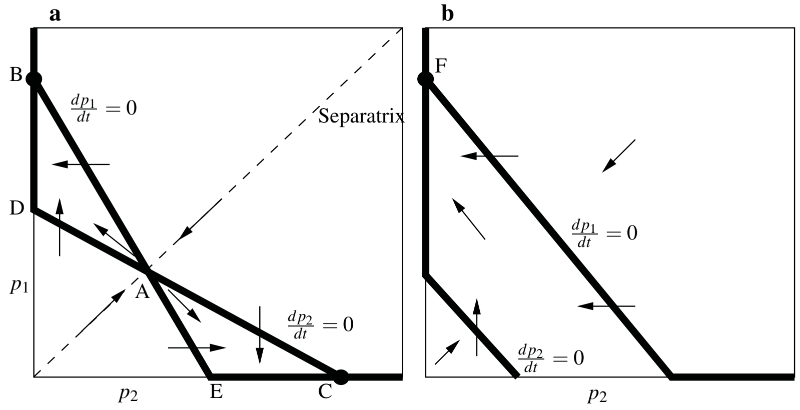


Fig. 4. The geometric structure of the flip-flop equations. Dark lines indicate the nullclines, where p_1 or p_2 is unchanging. The arrows indicate the sign of the derivatives at various regions in the phase space. **a** The hold state: a bistable system with balanced substrate concentrations. Points B and C are the stable points of the system. Point A is an unstable saddle node. **b** The reset state: a mono-stable system caused by a low concentration of substrate 2. The nullclines intersect once at point F, which all locations will be attracted to

The intersection at point A has four adjacent regions. Two of the regions adjacent to A have derivatives pointing toward A, but two of the regions have derivatives away from A. This reveals that A is a *saddle point*: perturbations in one direction will return back to A, while perturbations in another direction will fall away from A. B and C, on the other hand, are stable. The separatrix shown in Figure 4**a** divides the phase space into two basins of attraction. All trajectories starting above the separatrix will be attracted to B; all trajectories starting below it will be attracted to C. This configuration represents a hold state of the flip-flop as the system will evolve to either B or C. B is the low state of the flip-flop, in which the system has a low concentration of p_2 and a high concentration of p_1. Similarly, C is the high state, with a high concentration of p_2 and a low concentration of p_1. Thus we have two stable states and a valid flip-flop.

If the nullclines do not intersect in the positive quadrant then there will be a single intersection on either the p_1 or the p_2 axis. Such a situation is shown in Figure 4**b**. The point F is a stable node with a low p_2 value and a high p_1 value. An examination of the three regions of the phase space reveals that all points will be attracted to point F. Thus this state will reset the flip-flop. For bistability to occur the two nullclines must intersect in the orientation shown in Figure 4**a**. This requires that point B be above point D and that point C be to the right of point E, and translates into the following conditions: $kg_2 < r_1 g_1$ and $kg_1 < r_2 g_2$. If we wish to maintain symmetry, we can set $g_1 = g_2$ and $r_1 = r_2 = r$ and the conditions become $k < r$. When this condition is met the flip-flop will be in a hold state. The flip-flop is set or reset when the two nullclines do not intersect in the positive quadrant. Then there will exist one stable fixed point at their intersection (point F in Figure 4**b**). This will be the case if either r_1 or r_2 takes the value of zero. Suppose that the flip-flop is in the hold state with a high concentration of p_2 at point C in Figure 4**a**. As the parameter r_2 is reduced, the slope of the $\frac{dp_2}{dt}$ nullcline will decrease and the stable point C will slide down the p_2 axis toward point E. As C passes E the fixed point is annihilated, and the system will jump to the only fixed point: point F in Figure 4**b**. After the system has gone to point F the parameter r_2 can be returned to its original value and the flip-flop will remain in its low state. This lack of reversibility as the parameter is varied is called *hysteresis* in dynamical systems [7].

This stability analysis reveals the constraints we need to satisfy for the system to function as a flip-flop. First, in order to maintain output symmetry we require that $g_1 = g_2$. This ensures that both the output of the flip-flop (p_2) and the negated output (p_1) have the same value for a logical high. Next, in order to maintain a symmetric separatrix along the line $p_1 = p_2$ we require that $r_1 = r_2 = r$. Thus, during the hold state of the flip-flop, the phase-space is equally divided between values which are attracted to the high state and values which are attracted to the low state. Finally, the constraint for bistability is $k < r$. This means that the rate at which concentrations decay in the reactor (by means of outflow) must be less than the rate at which the enzymatic gates can create product. In other words, the gates must be capable of creating product faster than product is being removed.

We then convert these constraints on the dimensionless model to specifications for a physical system. Figure 5 shows a numerical integration of the system over a period of $9.6 \cdot 10^4$ s. At $2.4 \cdot 10^4$ s intervals the system is moved between set, hold, reset, and hold operations by controlling the influx of substrates. The top two traces on the left show

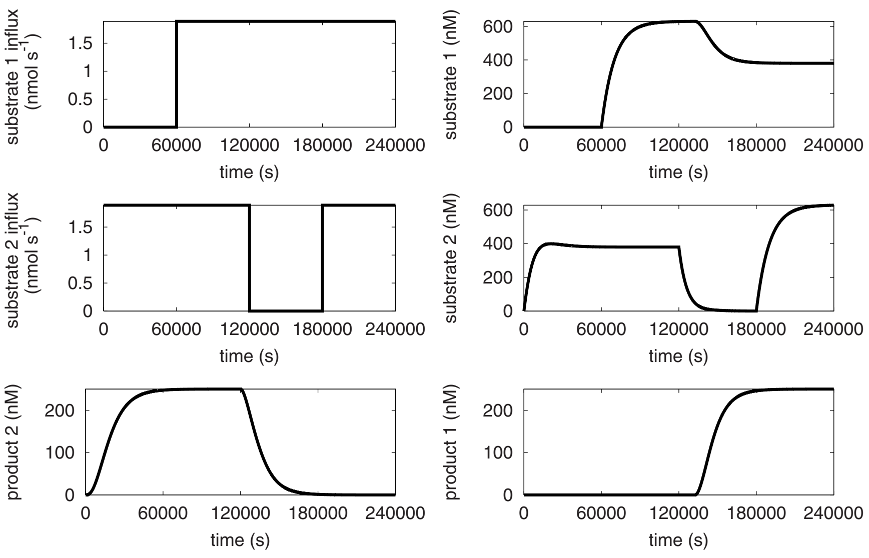


Fig. 5. Exercising control over the flip-flop system

substrate molecular influx rates. The top two traces on the right show the corresponding concentrations of substrates. The bottom two traces show concentrations of products 2 and 1. These represent the output and the negated output of the system, respectively.

6 Conclusions

Deoxyribozyme logic gates may be used to construct a biomolecular computer. By moving to open reactors, we increase the computational abilities of the underlying logic gates by making it possible to build recurrent circuits and devices with feedback. Techniques from dynamical systems offer qualitative and quantitative insights about the behavior of these chemical networks. Using these techniques, we have designed two fundamental components of a biomolecular computer: a single-bit inverter and a flip-flop that provides a single bit of memory. Compared to electronic computers, this technology is slow (about 1 mHz) but the possibility it offers of amorphous computation inside living cells is extremely exciting.

References

1. M. N. Stojanovic, T. E. Mitchell, and D. Stefanovic, “Deoxyribozyme-based logic gates,” *Journal of the American Chemical Society*, vol. 124, pp. 3555–3561, Apr. 2002.
2. M. N. Stojanovic and D. Stefanovic, “Deoxyribozyme-based half adder,” *Journal of the American Chemical Society*, vol. 125, pp. 6673–6676, May 2003.

3. M. N. Stojanovic and D. Stefanovic, “A deoxyribozyme-based molecular automaton,” *Nature Biotechnology*, vol. 21, pp. 1069–1074, September 2003.
4. M. Hiratsuka, T. Aoki, and T. Higuchi, “Enzyme transistor circuits for reaction-diffusion computing,” *IEEE Transactions on Circuits and systems—I: Fundamental Theory and Applications*, vol. 46, pp. 294–303, Feb. 1999.
5. I. R. Epstein and J. A. Pojman, *An Introduction to Nonlinear Chemical Dynamics*. New York: Oxford University Press, 1998.
6. C. Morgan, “Units of a biomolecular computer constructed from networks of modular chemical gates in an open reactor,” Undergraduate thesis, University of New Mexico, 2003.
7. S. H. Strogatz, *Nonlinear Dynamics and Chaos (With Applications to Physics, Biology, Chemistry, and Engineering*. Addison-Wesley, 1994.

Methods for Manipulating DNA Molecules in a Micrometer Scale Using Optical Techniques

Yusuke Ogura[1,3], Takashi Kawakami[1], Fumika Sumiyama[1,3], Satoru Irie[1], Akira Suyama[2,3], and Jun Tanida[1,3]

[1] Graduate School of Information Science and Technology, Osaka University, 2-1 Yamadaoka, Suita, Osaka 565-0871, Japan
{ogura, kawakami, sumiyama, irie, tanida}@ist.osaka-u.ac.jp
[2] Graduate School of Arts and Sciences, The University of Tokyo, 3-8-1 Komaba, Meguro-ku, Tokyo 153-8902, Japan
suyama@dna.c.u-tokyo.ac.jp
[3] Japan Science and Technology Corporation (JST-CREST)

Abstract. We studied optical methods for controlling the position and reaction of DNA molecules in a micrometer scale. We used an experimental system equipped with two single laser sources for optical manipulation and temperature control. The experimental results demonstrated that the position and reaction of the DNA were controlled independently. We also succeeded in manipulating DNA inside a microwell. The methods are expected to be useful for developing optically assisted DNA computing in which DNA is used as an information carrier and manipulated by effective use of optical techniques.

1 Introduction

Optical computing is a computational technique for parallel information processing that uses inherent property of light such as fast propagation, parallelism, and a large bandwidth. Many interesting results were obtained with various demonstration systems, which were associated with, for example, optical interconnection and digital optics[1, 2]. The embodiment of a valuable optical computing system requires high-performance optical devices, and some remarkable devices are developed. Good examples include vertical-cavity surface-emitting lasers (VCSELs)[3] and diffractive optical elements (DOEs)[4]. The VCSELs can be arranged in a two dimensional array structure with a period of a-few-hundred micrometer and their intensities are individually modulated at a rate that is higher than 10 GHz. Owing to the progress in a microfabrication technology based on lithography or other methods, DOEs that achieve complicated control of light are put to practical use. These devices are useful for controlling an optical field in a micrometer scale.

When using light, one must usually consider the diffraction limit, which determines the resolution of the light. From a viewpoint of information technology, the diffraction limit often restricts the density and capacity of information that is dealt with in a system. The precise alignment of optical devices is necessary for manipulating the light with the diffraction limit. The most important theme in the present research on optical computing is to find applications that utilize the full advantages of the features of the optical technique, and to contribute to develop practical and high performance information systems.

In contrast, DNA computing is an interesting computational paradigm in which information is encoded and expressed with the base sequences or the structures of DNA molecules[5]. Massive parallelism of reactions, smallness, capability to react autonomously, and other characteristics of DNA are effectively used in the DNA computing. Computation based on the DNA computing starts with design and synthesis of DNA. Information is processed by controlling reactions of the DNA. The DNA reacts in parallel and autonomously. This is an essential basis of the computational capability of the DNA computing. However, to use autonomous reactions of the DNA effectively, various information must be previously included in the base sequences of the DNA. Reaction parallelism suggests that a sophisticated mechanism or ingenuity is required for manipulating particular DNA molecules selectively in an entire DNA solution.

Light and DNA have large potential capabilities for achieving high performance information systems. There are, however, difficulties to be overcome at the present stage. Use of both the light and the DNA is a promising strategy to establish a new computational paradigm that has distinctive features. As an example, we propose optically assisted DNA computing, in which the DNA is used as an information carrier and manipulated by effective use of optical techniques. The optically assisted DNA computing is based on control of the DNA in a molecular scale by using bio-chemical reactions and that in a micrometer scale by applying optical techniques.

In this paper, we report on methods for controlling the position and reaction of DNA in a micrometer scale. We use a system equipped with two single laser sources of different wavelengths. Effectiveness of the methods is demonstrated by experimental results. The methods can be extended to parallel manipulation of DNA by using a VCSEL array[6] and therefore they are fundamental technologies for developing optically assisted DNA computing. In Section 2, the basic concept and the features of optically assisted DNA computing are described. In Section 3, methods for controlling the position and reaction of DNA by use of optical techniques are introduced. In Section 4, we show experimental results that are related with translation of DNA molecules by optical manipulation and local reaction control by irradiating with a laser beam.

2 Optically Assisted DNA Computing

The basic concept of optically assisted DNA computing is illustrated in Fig. 1. DNA molecules including encoded information, which are referred to as information DNAs in this paper, are used in the optically assisted DNA computing as well as in traditional DNA computing. The information DNAs are manipulated in a micrometer scale using optical techniques. This is a major difference

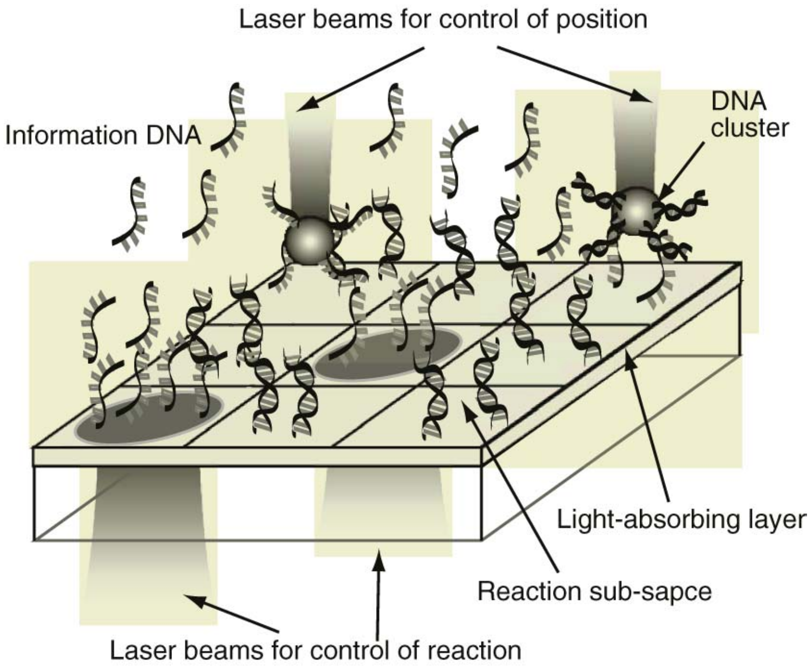

Laser beams for control of position
Information DNA
DNA cluster
Light-absorbing layer
Reaction sub-sapce
Laser beams for control of reaction

Fig. 1. The basic concept of optically assisted DNA computing

between the optically assisted DNA computing and the traditional DNA computing. The reaction space of DNA is divided into multiple small spaces, which are referred to as reaction sub-spaces. The reactions of information DNAs are controlled independently in the individual sub-spaces using optical techniques, and therefore particular information DNAs among all the information DNAs in the overall reaction space can be manipulated selectively. The information DNAs can be translated to another sub-space with microscopic beads. The information DNAs are attached to and detached from the substrate by hybridization and denaturation with the DNAs that are bound to the substrate. This makes possible to store the information DNAs. We can manipulate the information DNAs by generating and switching an optical field pattern.

The optically assisted DNA computing provides many advantages. From a viewpoint of the application of optical techniques to DNA computing, a procedural operation is introduced into the DNA computing owing to ease of controlling light. Use of the procedural operation is effective for relaxation of constraints in design of DNA sequences, improvement of flexibility in computation, and so on. An efficient processing or a new operation mechanism can be used because DNAs are manipulated in small reaction sub-spaces based on control from outside of the reaction space. An electronics technology can be combined to DNA computing because an optical technology bridges the both technologies. In addition, from a standpoint of optical computing, the capacity and density of information increase because the information can be manipulated in a molecular scale which is considerably smaller than the diffraction limit. Capability of DNA to react autonomously provides a large tolerance in controlling light and reduces requirements for system packaging. We can directly deal with ambiguous genetical information.

3 Manipulation of DNA Using Optical Techniques

Methods for manipulating information DNAs are important for developing optically assisted DNA computing. We present two methods using optical techniques. One is optical manipulation for controlling the position of the DNAs and the other is a method for controlling reaction of the DNAs in local space by laser irradiation.

Optical manipulation is a method for non-invasively manipulating an object with a radiation pressure force induced by the interaction between light and the object. Ashkin *et al.* have demonstrated optical levitation of a particle by a scattering force in 1971[7] and three-dimensional trapping of a microscopic dielectric particle by an optical gradient force in 1986[8]. The optical manipulation is useful for controlling a molecule, and it is applied to measure the physical property of a filament molecule and to clarify a mechanism of the motility of motor proteins[9, 10].

When DNA molecules are diffused in a solution, it is difficult to directly manipulate the DNA molecules by using a radiation pressure force. However, fabrication of a DNA cluster with a microscopic bead makes it possible to translate a large amount of DNAs by the optical manipulation. The DNA cluster is fabricated with a procedure as follows. Anti-tag DNAs that include a particular sequence are bound to the surface of a bead. Information DNAs include a tag sequence, which is complementary to the anti-tag DNA. Reacting with the bead, the information DNAs are immobilized to the bead by hybridization between the tag and anti-tag sequences. We refer to the bead with information DNAs as a DNA cluster. The DNA clusters are fabricated with the individual beads, and each DNA cluster contains thousands of the DNAs. A lot of information DNAs can be translated simultaneously by moving the DNA cluster.

Temperature is an essential parameter for control of various reactions of DNA molecules. Optical field patterns can be easily generated and modulated in a micrometer scale, and a temperature control method based on an optical technique is useful for controlling the DNA reactions locally. The scheme of the method for controlling reaction of DNAs by irradiating with a laser beam is shown in Fig. 2. A substrate is coated with material that absorbs light and a DNA solution is placed on it. When the substrate is irradiated with a focused laser beam, the surface of the substrate is heated up with light absorption. Thermal energy transfers to the solution on the substrate, and the temperature of the solution around irradiated area increases. Based on the phenomenon, the temperature of the solution can be controlled by changing the power of the beam used. When

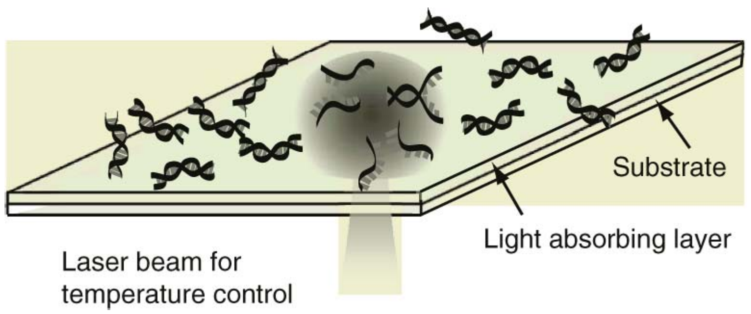


Fig. 2. A method for controlling the temperature of a solution in a micrometer scale

focusing a laser beam using an objective lens with a high numerical aperture, we can control a reaction of the DNAs selectively in a micrometer scale.

4 Experiments

Procedures to fabricate a DNA cluster and to prepare a substrate are as follows. Anti-tag DNAs (3'-GCACC TAGTC ATTGA CTTTA CTCCA TTCTA AACAT GATAC-5') which are modified with biotin at the 5'-end are mixed in a solution that contains polystyrene beads coated with streptavidin. The anti-tag DNAs are bound to the surfaces of the beads by biotin-streptavidin binding. Then tag DNAs (3'-GTATC ATGTT TAGAA TGGAG TAAAG TCAAT GACTA GGTGC-5') with fluorescent molecules (Molecular Probes: Alexa Fluor 647) for observation are mixed in the solution. The absorption and fluorescence emission maxima of the florescent molecules are 650 nm and 668 nm, respectively. After hybridization reaction, the tag DNAs are immobilized to the beads. The DNA clusters are extracted from the solution and a sample solution is prepared by putting DNA clusters into a buffer solution. A glass substrate is coated with titanylphthalocyanine of 0.15 μm thickness as a layer for absorbing light, then gold is deposited on the substrate. The anti-tag DNAs which are modified with thiol at the 5'-end are bound to the substrate.

We construct an experimental system equipped with two laser sources as shown in Fig. 3. The sample solution is irradiated from below with a laser beam (wavelength: 850 nm) for controlling temperature, whereas the solution is irradiated from above with a laser beam (wavelength: 980 nm) for positioning by optical manipulation. A dichroic mirror (Olympus: U-DM-CY5) is used for capturing fluorescent images. An excitation wavelength range is approximately 625 nm $\pm$ 25 nm.

The first experiment was detachment of information DNAs from the substrate. The information DNAs (tag DNAs) were immobilized to the substrate by hybridization with the anti-tag DNAs. We irradiated the substrate with a focused laser beam, and scanned the beam in the shape of the letter "T" by moving

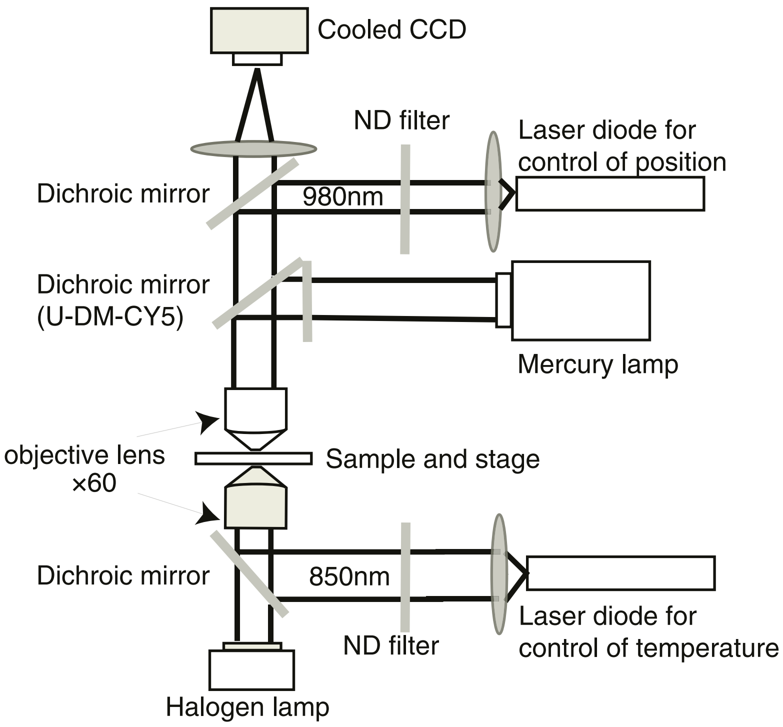


Fig. 3. Experimental setup

the sample stage. Figure 4 shows an fluorescent image captured after the above procedure. The fluorescent intensity of the area where the beam passed is lower than that of the other area. The phenomenon is explained by a mechanism that the information DNAs are denaturated by rise of temperature and diffused in the solution. The result shows that the denaturation of the information DNAs can be controlled by irradiating with a laser beam in approximately 5 μm resolution. After detaching the DNAs from the substrate, we performed a hybridization reaction with sufficient amounts of information DNAs. Then the fluorescent intensity at the area of detachment increased as high as that of surrounding area. The result suggests that the thiol-gold bond is not disrupted by laser irradiation.

We also performed an experiment for detaching information DNAs from a bead. In this experiment, we prepared two kinds of samples to measure photobleach of the fluorescent molecules. Sample (i) includes DNA clusters, in which fluorescent molecules are attached to the tag DNAs. Sample (ii) includes beads on which the anti-tag DNAs with fluorescent molecules are attached. We used a laser beam for temperature control and irradiated a bead in sample (i) or (ii) with 5 mW for 15 second then stopped irradiating for 4 second to capture a fluorescent image during one cycle. This irradiation cycle was repeated. Figure 5 shows dependence of the fluorescent power on an irradiation cycle. The decrease of the fluorescent power for sample (i) is considerably larger than that for sample (ii). In the case of sample (ii), the anti-tag DNAs are immobilized to the bead by biotin-streptavidin binding and do not leave by rise of temperature, and therefore the decrease of fluorescent power is considered as photobleach of the fluorescent molecules induced by the laser beam for temperature control and excitation light. This result demonstrates that information DNAs are detached from the bead by irradiating with the laser beam for temperature control.

We translated a DNA cluster and then denatured DNAs from the DNA cluster. The DNA clusters were fabricated with 6μm-diameter polystyrene beads. Figure 6 shows a sequence of fluorescent images during the experiment. (a) is the image at the initial state, (b) is that captured after translation of the DNA

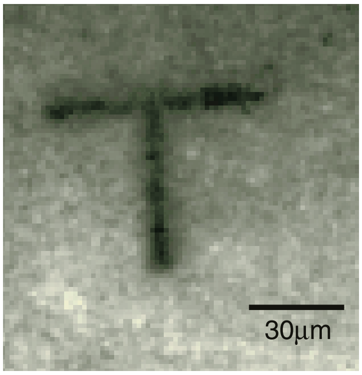


Fig. 4. Experimental result of detachment of information DNA from a substrate

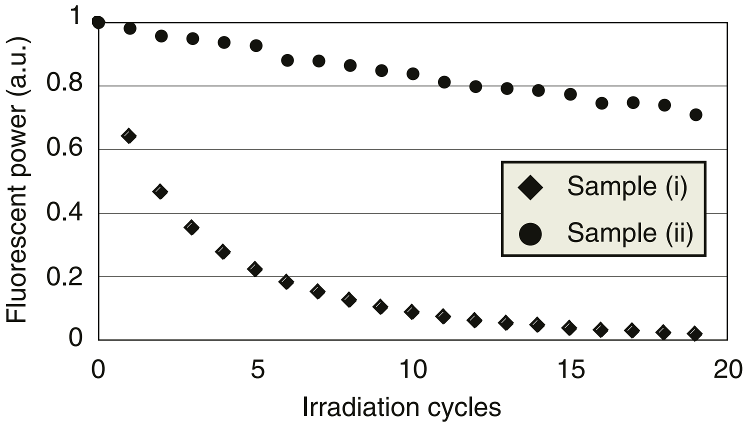


Fig. 5. Dependence of the fluorescent intensity on irradiation cycle when the laser beam for temperature control is used. Sample (i); fluorescent molecules are attached to tag DNAs, Sample (ii); fluorescent molecules are attached to anti-tag DNAs

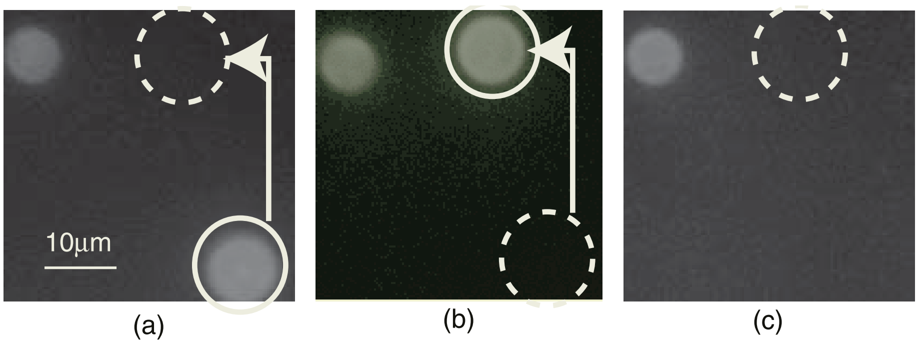


Fig. 6. Experimental result of translation and detachment of DNA. Fluorescent images (a) at initial state, (b) after translation, and (c) after detachment

cluster, and (c) is that after denaturation. For denaturation, we irradiated the DNA cluster with 3 mW for one minute. The fluorescent intensity clearly decreases after irradiating the DNA cluster with the laser beam for temperature control (Fig. 6(c)). In contrast, the fluorescent intensity scarcely decreases during irradiation with the laser beam for optical manipulation (Fig. 6(b)). This means that the DNAs are surely translated and then denatured. In addition, note that the fluorescent intensity of another DNA clusters does not decrease in Fig. 6. The experiment demonstrates that the position and reaction of DNAs can be controlled in a micrometer scale with our system.

To investigate a rate of denaturation with the laser beam for optical manipulation, we performed a further experiment with samples (i) and (ii) which are used in a previous experiment. We irradiated with the laser beam for optical manipulation for 15 second then stopped irradiating for 4 second to capture a fluorescent image during one cycle. This irradiation cycle was repeated. Little difference was seen between the fluorescent powers measured for samples (i) and (ii). This result suggests that the laser for optical manipulation can not increase temperature of the solution and it does not accelerate the denaturation reaction.

In addition to the above result, when the bead was irradiated with the laser beam for temperature control, the position of the DNA cluster was scarcely changed. The possible reasons are low transmittance to the substrate and spread of a focal spot by scattering in the substrate. These results show that the position and reaction of the DNA can be controlled independently using two wavelengths of laser sources.

A substrate with a microwell array was used in the following experiment. The microwell array is considered to be useful for confining information DNAs in subspaces. By controlling the reaction of the information DNAs inside the microwell, diffusion of them can be reduced. Furthermore, efficiency of the DNA reaction becomes higher due to the small volume of the reaction solution. Anti-tag DNAs are bound at the bottoms of the microwells, and therefore the information DNAs can be immobilized in the microwells when they include a tag sequence.

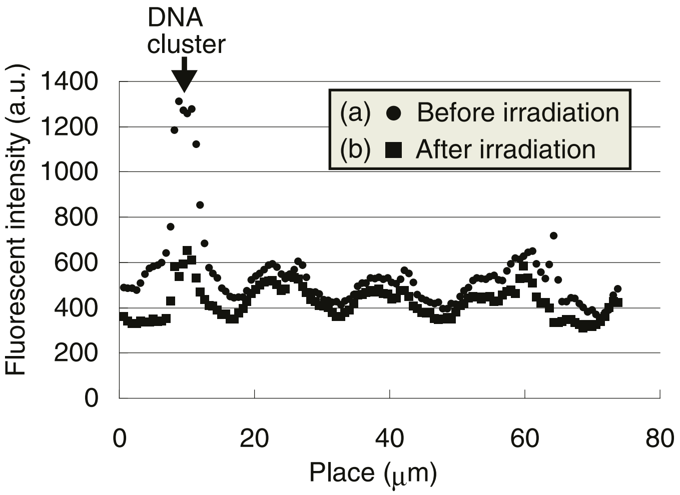


Fig. 7. Fluorescent intensity of a DNA cluster (a) before and (b) after irradiating with a laser beam

The microwell array was fabricated by photo-lithography and wet-etching processes. The diameter and the depth of the wells were 7 μm and 4 μm, respectively. A DNA cluster was put into a microwell by optical manipulation, and tag DNAs were denaturated by irradiating with a laser beam. The DNA clusters were fabricated with beads of 2μm in diameter. Fluorescent images of the DNA cluster were captured before and after irradiating with a laser beam. Dependence of the fluorescent intensity on the place along the diameter of the bead is shown in Fig. 7. The DNA cluster was irradiated with 4 mW for one minute. The fluorescent intensity decreases after irradiating with the laser beam. This result confirmed that the solution in the microwell was heated up and that the DNAs were denaturated due to a rise of temperature. We also verified that the DNA cluster in a microwell could be translated to anther microwell.

5 Conclusions

We developed methods for manipulating DNA molecules in a micrometer scale with optical techniques, and demonstrated the effectiveness of the methods. The experimental results confirmed that the position and reaction of DNA are controlled independently using two laser beams with different wavelengths. The methods can be applied to manipulate DNA in a microwell.

Measuring temperature when irradiating with a laser beam is an important issue in future. The result of Fig. 5 suggests that temperature can be increased higher than the melting temperature of DNA used (approximately 63℃) after several irradiation cycles. However, the accurate temperature have not been measured. Use of two sequences of DNA with different melting temperatures is a possible method to measure temperature.

Optically assisted DNA computing requires high efficient control of the reaction of DNA. For the purpose, use of a microwell array is a possible approach. The experimental result, however, suggests that the DNA molecules that leave from a bead are dissolved even though they react in the microwell. Closing the microwell with another bead is considered to be a strategy to prevent the DNA from dissolving. If the reaction space is completely sealed, the loss of the information DNA is avoidable, and therefore the DNA can be reacted with high efficiency. We are now examining this technique. The methods presented in this paper are useful in manipulating DNA in a micrometer scale and expected to be advanced as a new application of optics based techniques.

Acknowledgment

This work was supported by JST CREST and the Ministry of Education, Science, Sports, and Culture, Grant-in-Aid for Scientific Research (A), 15200023, 2003.

References

1. J. Tanida and Y. Ichioka: Digital optical computing. Progress in Optics **XL** (Elsevier Science, Amsterdam, 2000) 77-114.
2. J. Tanida and Y. Ichioka: Optical computing. The Optics Encyclopedia **3** (Wiley-VCH, Berlin, 2003) 1883-1902.
3. K. Iga: Surface-emitting laser – Its birth and generation of new optoelectronics field. IEEE J. Select. Topics in Quan. Electron. **6** (2000) 1201-1215.
4. H. P. Herzig (ed.): Micro-optics: Elements, systems, and applications. Taylor & Francis Ltd., 1 Gunpowder Square, London (1997).
5. J. Chen and J. Reif (eds.): Ninth Annual International Meeting on DNA based Computers, Preliminary proceedings, (2003).
6. Y. Ogura, T. Kawakami, F. Sumiyama, A. Suyama, and J. Tanida: Parallel translation of DNA clusters by VCSEL array trapping and temperature control with laser illumination. *Ninth Annual International Meeting on DNA based Computers* (2003) 19-27.
7. A. Ashkin and J. M. Dziedzic: Optical levitation by radiation pressure. Appl. Phys. Lett. **19** (1971) 283-285.
8. A. Ashkin, J. M. Dziedzic, J. E. Bjorkholm, and S. Chu: Observation of a single-beam gradient force optical trap for dielectric particles. Opt. Lett. **11** (1986) 288-290.
9. Y. Arai, R. Yasuda, K. Akashi, Y. Harada, H. Miyata, K. Kinosita Jr., and H. Itoh: Tying a molecular knot with optical tweezers. Nature **399** (1999) 446-448.
10. C. Veigel, L. M. Coluccio, J. D. Jontes, J. C. Sparrow, R. A. Milligan, and J. E. Molloy: The motor protein myosin-I produces its working stroke in two steps. Nature **398** (1999) 530-533.

From Cells to Computers: Membrane Computing – A Quick Overview

Gheorghe Păun

Institute of Mathematics of the Romanian Academy,
PO Box 1-764, 014700 Bucureşti, Romania
and
Research Group on Natural Computing,
Department of Computer Science and AI, University of Sevilla,
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
george.paun@imar.ro, gpaun@us.es

Abstract. The present paper is an informal introduction to membrane computing, presenting the basic ideas, the central (mathematical) results, and the main directions of research. A special emphasis is put on the (possible) usefulness of membrane systems for molecular computing and for biological modelling.

1 From Cells to Models

Membrane Computing (MC) started (in the end of 1998; the paper [12] was first circulated on web) from a double challenge: to check whether or not the statements about the computations taking place in a cell (see, e.g., [3], [9]) were mere metaphoras or they correspond to computations in the standard (mathematical) understanding of this term, and, more ambitiously, having in mind the encouraging experience of other branches of Natural Computing, to get inspired from the structure and the functioning of the living cell and define new computing models, possibly of interest for computer science, for computability in general.

In a very short time, the area developed extensively from a mathematical point of view, proposing and investigating a series of models (called P systems) inspired from the cell biochemistry. Later, also models inspired from the way the cells are organized in tissues and from the way the neurons cooperate in networks were considered. The main issues investigated were of a (theoretical) computer science type: computation power (in comparison with Turing machines and their restrictions), and usefulness in solving computationally hard problems. The field simply flourished at this level. Comprehensive information can be found in the web page (organized under the auspices of the European Molecular Computing Consortium, EMCC) [22]; a presentation at the level of the spring of year 2002 can be found in [13].

In short, a cell-like P system consists of a hierarchical arrangement of *membranes* (understood as three-dimensional vesicles), which delimit *compartments* (also called *regions*), where abstract *objects* are placed. These objects correspond to the chemicals from the compartments of a cell, and they can be either unstructured, a case when they can be represented by symbols from a given alphabet, or structured, and then a possible representation of objects is by strings over a given alphabet (but also more complex structures were considered, such as two-dimensional arrays, trees, etc). Corresponding to the situation from biology, where the number of molecules from a given compartment matters, also in the case of objects from the regions of a P system we have to take into account their multiplicity, that is why we consider *multisets* of objects assigned to the regions of P systems. These objects evolve according to *rules*, which are also associated with the regions. The intuition is that these rules correspond to the chemical reactions form cell compartments and the reaction conditions are specific to each compartment, hence the evolution rules are localized. The rules say both how the objects are changed and how they can be moved (we say *communicated*) across membranes. By using these rules, we can change the *configuration* of a system (the multisets from its compartments, and also the membrane structure in certain cases); we say that we get a *transition* among system configurations. The way the rules are applied imitates again the biochemistry (but goes one further step towards computability): the reactions are done in *parallel*, with the objects to evolve and the rules by which they evolve being chosen in a *non-deterministic* manner, in such a way that the application of rules is maximal. A sequence of transitions forms a *computation*, and with computations which *halt* (reach a configuration where no rule is applicable) we associate a *result*, for instance, in the form of the multiset of objects present in the halting configuration in a specified membrane.

The passage from the "real cell" to the "mathematical cell", as well as the generality of the approach are obvious. We start from the cell, but the abstract model deals with very general notions: membranes interpreted as separators of regions, objects and rules assigned to regions; the basic data structure is the multiset (a set with multiplicities associated with its elements); the rules are used in the non-deterministic maximally parallel manner, and in this way we get sequences of transitions, hence computations. All these items have a mathematical representation; in such terms, MC can be interpreted as a *bio-inspired framework for distributed parallel processing of multisets.*

It is also obvious that from a theoretical point of view, MC is an extension of DNA (more generally, of Molecular) Computing, both because the objects from a P system can be DNA molecules, processed by operations abstracting from DNA biochemistry (actually, there were considered splicing based P systems, and other P systems with string-objects which evolve according to biochemical-like rules), and because MC deals with a distributed parallel computing, in a cell-like framework, which is a model of distributed DNA Computing both *in vivo* and *in vitro.*

Thus, MC started as a mathematical enterprise inspired from biology, not aiming to model the cell in the benefit of cell biology, but in the benefit of computability. Several seemingly contradictory criteria have concomitantly acted: to have models as realistic as possible (adequate to the bio-reality, capturing as many biological features as possible), as powerful as possible from a computability point of view (if possible, equal in power to Turing machines; this is rather important, because the constructive equivalence with Turing machines can directly imply the existence of universal – hence "programmable" – computing models), as elegant as possible from a mathematical point of view (with reduced ingredients, without redundancies, etc), as efficient as possible (solving computationally hard problems in a feasible time, by making use of the parallelism inherent to the biochemistry of the cell). Some details about these criteria will be given below. Important is to note that among the many classes of P systems considered in the literature, many of them incorporate in a direct way various biologically inspired features, most of them are Turing universal, even in cases where reduced ingredients are used (which is a way to get farther from biology, where one meets a lot of redundancies), while several classes of P systems can provide polynomial or even linear time solutions to **NP**-complete problems (by a time-space trade-off, made possible, for instance, by membrane division). Otherwise stated, at the theoretical level, MC was quite successful. Based on this mathematical development, two recent trends are rather promising also from a practical point of view. On the one hand, one discusses more and more frequently about the usefulness of MC as a source of ideas for distributed computing and as a possible general framework for covering and extending several existing approaches to distributed computing. This direction of research, still in its infancy, is directly continuing the many efforts for implementing (simulating) P systems on the usual electronic computer (there are numerous programs of this type, see [22]), on distributed architectures [6], [20], on a reconfigurable hardware [16], etc. On the other hand, making use of both the theoretical developments and of the existing implementations, MC started to be used as a framework for modelling biological processes, especially at the level of the cell. This was not an initial goal of MC, but it promises to be an important "by-product" of it, in view of the difficulty of modelling and simulating the cell – see, e.g., [21]. We will return to these topics, after having a closer look to MC.

2 The Basic Classes of P Systems

We introduce now the fundamental ideas of MC in a more precise way. We recall the fact that, wearing "mathematical glasses", we look to the cell and try to find a computing device. To this aim we need *data structures* and *operations* with these data structures, an *architecture* of our "computer", and a systematic manner to define *computations* and *results* of computations. As a result, MC proposes a computing model in the form of a *membrane system* (or *P system*), consisting of (1) a *membrane structure*, in whose compartments there are placed (2) *multisets* of (3) *objects* which evolve according to (4) sets of *rules* also associated with the compartments.

A membrane structure is a hierarchically arranged set of membranes. A suggestive representation is as in Figure 1. We distinguish the external membrane (corresponding to the plasma membrane and usually called the *skin* membrane) and several internal membranes (corresponding to the membranes present in a cell, around the nucleus, in Golgi apparatus, vesicles, etc); a membrane without any other membrane inside it is said to be *elementary*. Each membrane determines a compartment, also called *region*, the space delimited from above by it and from below by the membranes placed directly inside, if any exists. The correspondence membrane–region is one-to-one, that is why we sometimes use interchangeably these terms; also, we identify by the same label a membrane and its associated region.

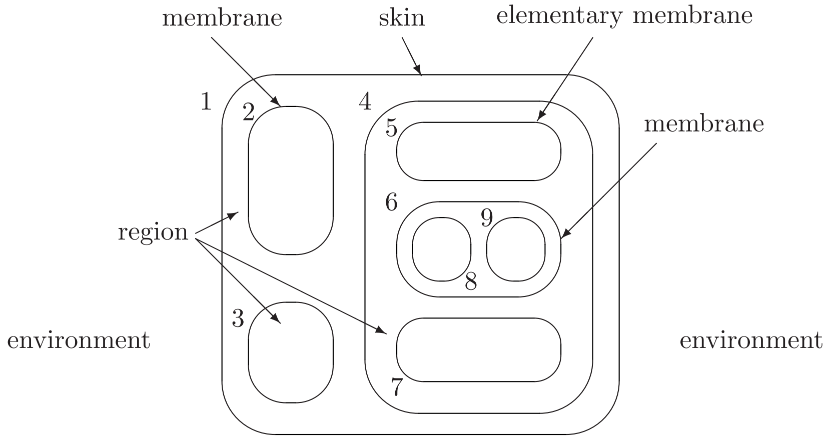


Fig. 1. A membrane structure

In the basic class of P systems, each region contains a multiset of symbol-objects, which correspond to the chemicals swimming in a solution in a cell compartment; these chemicals are considered here as unstructured, that is why we describe them by symbols from a given alphabet.

The objects evolve by means of evolution rules, which are also localized, associated with the regions of the membrane structure. The rules correspond to the chemical reactions possible in the compartments of a cell. The typical form of such a rule is $aad \rightarrow (a, here)(b, out)(b, in)$, with the following meaning: two copies of object a and one copy of object d react and the reaction produces one copy of a and two copies of b; the new copy of a remains in the same region (indication *here*), one of the copies of b exits the compartment, going to the surrounding region (indication *out*) and the other enters one of the directly inner membranes (indication *in*). We say that the objects a, b, b are *communicated* as indicated by the commands associated with them in the right hand member of the rule. When an object exits a membrane, it will go to the surrounding compartment; in the case of the skin membrane this is the environment, hence the object is "lost", it never comes back into the system. If no inner membrane exists (that is, the rule is associated with an elementary membrane), then the indication *in* cannot be followed, and the rule cannot be applied.

The communication of objects through membranes reminds the fact that the biological membranes contain various (protein) channels through which the molecules can pass (in a passive way, due to concentration difference, or in an active way, with a consumption of energy), in a rather selective manner. The fact that the communication of objects from a compartment to a neighboring compartment is controlled by the "reaction rules" is attractive mathematically, but not quite realistic from a biological point of view, that is why there also were considered other ways of moving objects, for instance, by symport/antiport rules – see Section 3.

A rule as above, with several objects in its left hand member, is said to be *cooperative*; a particular case is that of *catalytic* rules, of the form $ca \rightarrow cx$, where a is an object and c is a catalyst, appearing only in such rules, never changing. A rule of the form $a \rightarrow x$, where a is an object, is called *non-cooperative*.

The rules associated with a compartment are applied to the objects from that compartment, in a *maximally parallel way*: we assign objects to rules, until no further assignment is possible. The used objects are "consumed", the newly produced objects are placed in the compartments of the membrane structure according to the communication commands assigned to them. The rules to be used and the objects to evolve are chosen in a non-deterministic manner. All compartments of the system evolve at the same time, synchronously (a common clock is assumed for all membranes). Thus, we have two layers of parallelism, one at the level of compartments and one at the level of the whole "cell".

A membrane structure and the multisets of objects from its compartments identify a *configuration* of a P system. By a non-deterministic maximally parallel use of rules as suggested above we pass to another configuration; such a step is called a *transition*. A sequence of transitions constitutes a *computation*. A computation is successful if it halts, it reaches a configuration where no rule can be applied to the existing objects. With a halting computation we can associate a *result* in various ways. The simplest possibility is to count the objects present in the halting configuration in a specified elementary membrane; this is called *internal output*. We can also count the objects which leave the system during the computation, and this is called *external output*. In both cases the result is a number. If we distinguish among different objects, then we can have as the result a vector of natural numbers. The objects which leave the system can also be arranged in a sequence according to the moments when they exit the skin membrane, and in this case the result is a string. A string can also be obtained by following the *trace* of a distinguished object (a "traveller") through membranes. (When we deal with string-objects, then the result of a computation is usually defined as the set of strings which leave the system during the computation; these strings cannot be further processed, hence in this case we do not need to use the halting condition in order to define successful computations.)

Because of the non-determinism of the application of rules, starting from an initial configuration, we can get several successful computations, hence several results. Thus, a P system *computes* (one also uses to say *generates*) a set of numbers, or a set of vectors of numbers, or a language.

Of course, the previous way of using the rules from the regions of a P system reminds the non-determinism and the (partial) parallelism from cell compartments, with the mentioning that the maximality of parallelism is mathematically oriented (rather useful in proofs); when using P systems as biological models, this feature should be replaced with more realistic features (e.g., reaction rates, probabilities, partial parallelism).

An important way to use a P system is the automata-like one: an *input* is introduced in a given region and this input is *accepted* if and only if the computation halts. This is the way of using P systems, for instance, in solving decidability problems.

We do not give here a formal definition of a P system. The reader interested in mathematical and bibliographical details can consult the mentioned monograph [13], as well as the relevant papers from the web bibliography [22]. Of course, when presenting a P system we have to specify: the alphabet of objects (a usual finite non-empty alphabet of abstract symbols identifying the objects), the membrane structure (usually represented by a string of labelled matching parentheses), the multisets of objects present in each region of the system (represented by strings of symbol-objects, with the number of occurrences of a symbol in a string being the multiplicity of the object identified by that symbol in the multiset represented by the considered string), the sets of evolution rules associated with each region, as well as the way the output is defined.

Many modifications/extensions of the very basic model sketched above are discussed in the literature, but we do not mention them here.

3 Computing by Communication

In the systems described above, the symbol-objects were processed by multiset rewriting-like rules, and moved across membranes according to given communication targets. Coming closer to the trans-membrane transfer of molecules, we can consider purely communicative systems, based on the two classes of coupled transfer processes known in biology: *symport*, and *antiport* (see [1] for details). Symport refers to the transport where two (or more) molecules pass together through a membrane in the same direction, antiport refers to the transport where two (or more) molecules pass through a membrane simultaneously, but in opposite directions; for uniformity, the case when a molecule does not need a "partner" for a passage is referred to as uniport.

In terms of P systems, we can consider object processing rules of the following forms: a symport rule (associated with a membrane i) is of the form (x, in) or (x, out), stating that the objects of the multiset (represented by the string) x enter/exit together membrane i, while an antiport rule is of the form $(x, out; y, in)$, stating that, simultaneously, the objects from x exit and those from y enter membrane i.

A P system with symport/antiport rules has the same architecture and functioning as a system with multiset rewriting rules: alphabet of objects, membrane structure, initial multisets in the regions of the membrane structure, sets of rules associated with the membranes, possibly an output membrane – with one additional component, the set of objects present in the environment. Because by communication we do not create new objects, we need a supply of objects, in the environment, otherwise we are only able to handle a finite population of objects, those provided in the initial multiset. If an object is present in the environment at the beginning of a computation, then it is considered available in arbitrarily many copies (the environment is inexhaustible). In this way, the environment takes an active part in the computation, which, together with the conservation of objects, the direct biological inspiration, the mathematical elegance, the computational power, is one of the attractive features of this class of P systems.

4 Computational Completeness; Universality

As we have mentioned before, many classes of P systems, combining various ingredients biologically or mathematically inspired, are able of simulating Turing machines, hence they are *computationally complete*. Always, the proofs of results of this type are constructive, and this has an important consequence from the computability point of view: there are *universal* (hence *programmable*) P systems. In short, starting from a universal Turing machine (or an equivalent universal device), we get an equivalent universal P system. Among others, this implies that in the case of Turing complete classes of P systems, the hierarchy on the number of membranes always collapses (at most at the level of the universal P systems). Actually, the number of membranes sufficient in order to characterize the power of Turing machines by means of P systems is always rather small.

We only mention here two of the most interesting universality results:

1. P systems with symbol-objects with catalytic rules, using only two catalysts and two membranes, are computationally universal.
2. P systems with symport/antiport rules of a rather restricted size (example: four membranes, symport rules of weight 2, and no antiport rules) are universal.

We can conclude that the compartmental computation in a cell-like membrane structure (using various ways of communicating among compartments) is rather powerful. The “computing cell” is a powerful “computer”.

5 Computational Efficiency

The computational power (the “competence”) is only one of the important questions to be dealt with when defining a new computing model. The other fundamental question concerns the computing *efficiency*. Because P systems are parallel computing devices, it is expected that they can solve hard problems in an efficient manner – and this expectation is confirmed for systems provided with ways for producing an exponential workspace in a linear way. Three main such possibilities have been considered so far in the literature, and *all of them were proven to lead to polynomial solutions to* **NP***-complete problems.*

These three ideas are *membrane division*, *membrane creation*, and *string replication.* Using these operations (especially membrane division) polynomial time solutions to SAT, the Hamiltonian Path problem, the Node Covering problem, the problem of inverting one-way functions, the Subset-sum and the Knapsack problems (note that the last two are numerical problems, where the answer is not of the yes/no type, as in decidability problems), etc were proposed. Details can be found in [13], [15], as well as in the web page of the domain.

Roughly speaking, the framework for dealing with complexity matters is that of *accepting P systems with input*: a family of P systems of a given type is constructed starting from a given problem, and an instance of the problem is introduced as an input in such systems; working in a deterministic mode (or a *confluent* mode: some non-determinism is allowed, provided that the branching converges after a while to a unique configuration), in a given time one of the answers yes/no is obtained, in the form of specific objects sent to the environment.

This direction of research is very active at the present moment. More and more problems are considered, the MC complexity classes are refined, characterizations of the $\mathbf{P}\neq\mathbf{NP}$ conjecture were obtained in this framework, improvements are looked for. An important recent result states the fact that **PSPACE** is included in $\mathbf{PMC}_D$, the family of problems which can be solved in polynomial time by P systems with the possibility of dividing both elementary and non-elementary membranes [17]. Rather interesting are the investigations on the complexity of various problems which are known to be decidable (see, e.g., [8]).

6 Applications

As said several times in the previous sections, MC was initiated with the goal of finding ideas, models, tools of interest for computer science in the cell structure and functioning, and not of modelling the real cell. However, abstracting from the cell biochemistry, a new modelling framework (starting with a new language, set of concepts, tools) was developed which proves now to be useful for modelling not only biological processes, but also linguistic facts, management aspects, etc. Several applications in addressing computer science problems were reported, for instance, in sorting and ranking problems, handling 2D languages, in computer graphics, in simulating Fredkin gates or shuffle-exchange networks, etc. A book is in preparation, covering many of these applications.

In some of these applications, what is actually used is the *language* of MC. This means not only the long list of concepts either newly introduced, or related in a new manner in this area, but also the way to represent a cell-like structure, as proposed in MC. This representation is rather attractive for biologist: Euler-Venn diagrams, with labels for membranes, with multisets of objects (chemicals) placed in regions, and with sets of rules placed either in regions (the case of rewriting-like rules) or near membranes, to suggest that they are associated with the membranes (the case of symport/antiport rules).

However, this level of application/usefulness is only a preliminary one. The next level is that of using tools, techniques, results of MC, and here there appears an important question: to which aim? Solving problems already stated by biologists, in other terms and another framework, could be an impressive achievement, and this is the most natural way to proceed – but not necessarily the most efficient one, at least at the beginning. New tools can suggest new problems, which either cannot be formulated in a previous framework or have no chance to be solved in the previous framework. Problems of the first type (already examined by biologists, mainly experimentally) concern, for instance, correlations of processes, of the presence/absence of certain chemicals, their multiplicity (concentration, population) in a given compartment, while of the second type are topics related to the evolution of bio-systems when modelled as dynamical systems (e.g., concerning the periodicity and the asymptotic properties of trajectories, or the reachability of certain configurations).

The applications in biology develop in general along a scenario of the following type: one examines a piece of reality, related to the biochemistry of the cell (but not only from this area), one builds a P system modelling the respective process, one writes a program simulating that system (or one uses one of the many existing programs), and one performs experiments with the program, tunning certain parameters, and looking for the evolution of the system (usually, for the population of certain objects). Respiration in bacteria [2], photosynthesis [11], processes related to the imune system [7], [19], and other processes [5], [18] were studied in this way. We do not recall any detail here, but we refer to the papers mentioned above, to the chapter of [13] devoted to biological applications, as well as to the papers available in [22]. Anyway, the investigations are somewhat preliminary, but the progresses are obvious and the expectation is to have in the near future applications of an increased interest for biologists. This hope is supported also by the fact that more and more powerful simulations/implementations of various classes of P systems are available, with better interfaces, which allow for the friendly interaction with the program.

7 Why Membrane Computing?

This is a rethorical question, but still we take and answer it here in a serious manner.

The cell exists, so it is the “duty” of mathematics to model it – especially having in mind that the biologists are waiting for mathematical tools to help their investigations, in particular, they wait for global models of the cell. And now, comes an important question: using what kind of mathematics? Traditionally, the mathematical models are equalized with the continuous mathematical models, especially (partial) differential equations – and this is the case also with biology, which tries to use such models since several years. The results are only partially satisfactory: the cell is too complex in order to be modelled as a whole, while the differential equations are difficult to handle, difficult to understand by biologist, difficult to scale-up, and so on and so forth. A similar situation occurred in linguistics about half a century ago, when it was soon realized that the right approach is not the one based on the continuous mathematics, but the discrete algorithmic models. The biology has one more peculiarity: immense databases were accumulated (see, e.g., the genome project), which are waiting to be explored. The need to use computers is obvious. Moreover, most of these biological data are of a discrete type. (This is another topic, which we only mention here: how is the *real* biological reality, of a continuous or of a discrete type, is it a space-time field, or it deals mainly with "quanta" of matter?) MC proposes and investigates models which are both discrete and algorithmic, two attractive and timely considered features. Using discrete models in approaching the cell as a whole was several times advocated, due to the easy understandability for the biologist, the easy scalability, the efficiency in simulating on computers.

Then, if not for other reasons, P systems deserve to be considered as possible cell models at least for the obvious one that this is the first class of models directly inspired from cell structure and functioning, while the tools previously used in cell modelling were borrowed from other areas, from physics and computer science in general, and needed considerable efforts to be adjusted to the biological reality.

But, let us return to the initial goal of MC, that also related to DNA Computing. Why MC as a branch of computer science? At a previous DNA Computing conference, somebody has formulated it much more precisely: what can MC do better than other branches of computer science? Again the answer can be given at various levels. First, the cell is such an exquisite tiny machinery, polished by nature for billions of years, that there is a good chance to find in it, in its architecture and way of processing substances and information, various ideas, techniques, models also useful to computer science. The case of other branches of Natural Computing (see, e.g., evolutionary computing) is rather encouraging. MC is a systematic investigation of computing models inspired from the cell structure and functioning, and there is no other branch of computer science with similar goals. In particular – and in contrast to DNA Computing – MC addresses in a direct manner the compartmentalization of the cell.

This is a crucial point: the distribution/parallelism of computations which take place in P systems. The parallelism is a long dream of computer science (often, both Turing and von Neumann are "blamed" for pushing computer science towards sequential computing). Nature "computes" almost always in parallel, from cell compartments, to tissues, brain, organisms, populations. From a technological point of view, there is no difficulty in putting together a huge number of electronic processors/computers (internet is an example); the difficulty is to use them, to coordinate, synchronize, etc (in distributed/parallel computing, the communication complexity is equally important and contradictory to classic time and space complexities). Nature has solved this difficulty in a way which remains to be discovered and incorporated in our "artificial" computing machineries. Decentralization, random choices, non-determinism, loose control, asynchronization, promoting/inhibiting, control paths are key-words of the "natural computing", of the manner the cell preserves its integrity, its life, but they are still long term goals for computer science *in silico*, a challenge for computability in general.

We do not claim that MC provides an answer to this challenge, but it is at least a framework to formulate this question and to (re)call the attention to it.

At a much more technical level, MC proved its attractivity for computer science by the two types of results which are common in this area: Turing universality and fast solutions to hard problems. The borderline between universality and non-universality is a central topic in computability, in many cases a non-trivial issue. The cell, abstracted in MC terms, turned out to be a very powerful "computer", universal in many cases, including the surprising and philosophically significant case of computing by communication only, in the case of symport/antiport P systems. Then, numerous efforts were paid to the complexity classes corresponding to various types of P systems where an exponential space can be created in a linear time (e.g., by membrane division, a well-know biological operation). The "standard" result is a proof that a given **NP**-complete problem can be solved in polynomial or even linear time by P systems of a given type, but more interesting results were also reported. The possibility to cover **PSPACE** and to characterize the $\mathbf{P} \neq \mathbf{NP}$ problem were already mentioned. Interesting questions appear in what concerns the very definition of a complexity class in this framework. We have mentioned in Section 5 the possibility of allowing some non-determinism in the functioning of systems which are meant to solve a decision problem. Another delicate question concerns the way of constructing the systems which solve a problem. Classicaly, these systems should be constructed in a uniform mode (starting from the size of instances) by a Turing machine, working a polynomial time. A more relaxed framework is that where a *semi-uniform* construction is allowed: carried out in polynomial time by a Turing machine, but starting from the instance itself to be solved (the condition to work a polynomial time ensures the "honesty" of the construction: the solution to the problem cannot be found during the construction phase). The relation between the complexity classes based on uniform and semi-uniform constructions is not clarified yet. Then, several challenging hypotheses circulate in the area, about the features necessary in order to obtain a given degree of computational efficiency. For instance, in [17] it was formulated the conjecture that in order to solve `QSAT` in polynomial time by means of P systems with active membranes, the division of both elementary and non-elementary membranes is necessary (when solving, e.g., `SAT` in linear time, division of elementary membranes was sufficient).

8 Concluding Remarks

This was a very general overview of membrane computing, informal, covering only a few directions of research and giving very few bibliographical references. The aim was to give the reader a quick image of the area, pointing the basic ideas, the main types of results (universaliry and efficiency), the important recent trends (towards applications in computer science and in biological modelling), and to emphasize the interest for this research (both through the possible achievements and through the attractive research topics). The reader interested in more details is advised to consult the comprehensive bibliography from [22].

References

1. B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, P. Walter, *Molecular Biology of the Cell*, 4th ed. Garland Science, New York, 2002.
2. I.I. Ardelean, M. Cavaliere, Modelling Biological Processes by Using a Probabilistic P System Software. *Natural Computing*, 2, 2 (2003), 173–197.
3. D. Bray, Protein Molecules as Computational Elements in Living Cells. *Nature*, 376 (July 1995), 307–312.
4. C.S. Calude, Gh. Păun, G. Rozenberg, A. Salomaa, *Multiset Processing. Mathematical, Computer Science, and Molecular Computing Points of View. Lecture Notes in Computer Science*, 2235, Springer, Berlin, 2001.
5. G. Ciobanu, D. Dumitru, D. Huzum, G. Moruz, B. Tanasă, Client-Server P Systems in Modelling Molecular Interaction. In [14], 203–218.
6. G. Ciobanu, W. Guo, P Systems Running on a Cluster of Computers. In [10], 123–139.
7. G. Franco, V. Manca, A Membrane System for the Leukocyte Selective Recruitment. In [10], 180–189.
8. O.H. Ibarra, On the Computational Complexity of Membrane Computing Systems. *Theoretical Computer Science*, 320, 1 (2004), 98–109.
9. S. Ji, The Cell as the Smallest DNA-based Molecular Computer. *BioSystems*, 52 (1999), 123–133.
10. C. Martin-Vide, G. Mauri, Gh. Păun, G. Rozenberg, A. Salomaa, eds. *Membrane Computing. International Workshop, WMC2003, Tarragona, Spain, Revised Papers. Lecture Notes in Computer Science*, 2933, Springer, Berlin, 2004.
11. T.Y. Nishida, Simulations of Photosynthesis by a K-subset Transforming System with Membranes. *Fundamenta Informaticae*, 49, 1-3 (2002), 249–259.
12. Gh. Păun, Computing with Membranes. *Journal of Computer and System Sciences*, 61, 1 (2000), 108–143 (and Turku Center for Computer Science-TUCS Report 208, November 1998, `www.tucs.fi`).
13. Gh. Păun, *Computing with Membranes: An Introduction.* Springer, Berlin, 2002.
14. Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, *Membrane Computing. International Workshop, WMC-CdeA 2002, Curtea de Argeş, Romania, Revised Papers. Lecture Notes in Computer Science*, 2597, Springer, Berlin, 2002.
15. M. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrini, *Teoría de la Complejidad en Modelos de Computatión Celular con Membranas*, Editorial Kronos, Sevilla, 2002.
16. B. Petreska, C. Teuscher, A Hardware Membrane System. In [10], 269–285.
17. P. Sosik, The Computational Power of Cell Division in P Systems: Beating Down Parallel Computers? *Natural Computing*, 2, 3 (2003), 287–298.
18. Y. Suzuki, Y. Fujiwara, H. Tanaka, J. Takabayashi, Artificial Life Applications of a Class of P Systems: Abstract Rewriting Systems on Multisets. In [4], 299–346.
19. Y. Suzuki, S. Ogishima, H. Tanaka, Modeling the p53 Signaling Network by Using P Systems. *Proceedings of the Brainstorming Week on Membrane Computing; Tarragona, February 2003* (M. Cavaliere, C. Martin-Vide, Gh. Păun, eds.), Technical Report 26/03, Rovira i Virgili University, Tarragona, 449–454.

20. A. Syropoulos, E.G. Mamatas, P.C. Allilomes, K.T. Sotiriades, A Distributed Simulation of Transition P Systems. In [10], 357–368.
21. M. Tomita, Whole-Cell Simulation: A Grand Challenge of the 21st Century. *Trends in Biotechnology*, 19 (2001), 205–210.
22. * * *, P Systems Web Page: `http://psystems.disco.unimib.it`.

The Capacity of DNA for Information Encoding

Vinhthuy Phan and Max H. Garzon

Computer Science, The University of Memphis,
Memphis, TN 38152-3240, U.S.A.
{vphan, mgarzon}@memphis.edu

Abstract. Information encoding and processing in DNA has proved to be an important problem for biomolecular computing, including the well studied codeword design problem. A lower bound is established for the capacity of DNA to encode information using a combinatorial model of DNA homology given by the so-called h-distance. This bound decreases exponentially with a parameter τ that roughly codes for stringency in reaction conditions. We further introduce a new family of near-optimal codeword sets, so-called *shuffle* codes. This construction, which is optimal in terms of efficiency, can also be used to produce set of codewords with a given constant GC-content. These codes yield estimates of the capacity of DNA oligonucleotides to store abiotic information in DNA arrays as defined in [11]. Finally, we discuss the sensitivity of the corresponding DNA chip encodings to store and discriminate inputs, including the regions of maximum discrimination and uncertainty.

Keywords: Information storage capacity of DNA, Gibbs energy, h-distance, codeword design, shuffle codes, abiotic data on DNA chips.

1 Introduction

The ways cells and biomaterials store and process information is one of the most important and enigmatic problems of our times. One major subproblem of this puzzle, encoding information in biomolecules for processing *in vitro*, has proven to be a interesting and challenging problem for biomolecule-based computing after Adelman's founding work [1]. Codeword design and, more generally, data and information representation in DNA bear an increasing interest, not only from the point of view of using biomolecules for computation, but also for shedding light on a number of other problems in areas outside computation *per se*, such as bioinformatics, and conceivable genetics and microbiology. The *codeword design problem* [4, 5, 8, 11, 12, 13] requires producing sets of strands that are likely to bind in desirable hybridizations while keeping to a minimum the probability of erroneous hybridizations that may induce false positive or false negative outcomes. A fairly extensive literature now exists on various aspects and approaches to the problem (see [5] for a review.) Although various algorithms have been proposed for testing the quality of codeword sets in terms of being free of secondary structure [5], very few methods have been proposed to *systematically* produce codes of high enough quality to guarantee good performance in test tube protocols [10]. Even fewer articles address the issue of the capacity of DNA strands of a given length (say 20−mers) to hold information, if they are to co-exist without undesirable cross interactions in a test tube under given stringency conditions, such as required by a stable associative memory "in principle larger than the brain" [4].

The obvious approach to encode information in DNA is to encode bits 0 and 1 by fixed selected oligos and symbolic strings by the induced concatenations. Direct encoding is, however, not very efficient for storage or processing of massive amounts (over terabytes) of abiotic data because of the enormous implicit cost of DNA synthesis to produce the encoding strands. Indirect and more efficient methods have been proposed [10], using as basic elements a set of non-cross-hybridizing DNA molecules. DNA molecules can interact through intermolecular reactions, usually hybridization in DNA form alone. Due to the inherent uncertainty in biochemical processes, small variations in strand composition will not cause major changes in hybridization events, with consequent limitations on using small variations of a molecule to encode different records. Input strands must be "far apart" from one another in hybridization affinity in order to prevent disturbing cross-hybridizations. The major difficulty is that the hybridization affinity between DNA strand sets is difficult to quantify. Ideally, the Gibbs energy released in the process should be the most appropriate criterion, but its exact calculation is difficult. Even though pairwise interactions among small oligos may admit fast approximations [7], a search for optimal *sets* of strands maximally separated in a given coding space (codeword sets) is infeasible, because the optimality of a set is a global property to be checked against doubly exponentially many others possibilities. In particular, the capacity of DNA oligonucleotides of a given length to store information in the globally robust and fault-tolerant way that biology appears to require seems to be very difficult to quantify, or even estimate.

To cope with this problem, a much simpler and computationally tractable model of the Gibbs energy landscapes given by the hybridization distance (or just h-distance), was introduced in [12] as a measure of hybridization affinity. In this approximation, a space of DNA strands becomes a combinatorial object (a string) over a 4-letter alphabet in which characters may "bond" according to the well known Watson-Crick pairing (A-T's and C-G's). The Gibbs energy is replaced, not by the traditional Hamming distance assuming perfect alignment between the strings in simpler models [15, 16], but by a value that counts the largest number of matching pairs in the most favorable relative shift as the strands slide past each other rigidly, unable to form loops or bulges. Hybridization events, naturally dependent on many other reactions conditions such as temperature, salinity, and kinetic factors, are reduced to a single numerical threshold τ. Hybridization occurs if and only if their h-distance does not exceed τ (more below in Section 2.) For example, under the tightest stringency $\tau = 0$, two strings can only hybridize when they are perfect complements, whereas under the most relaxed stringency $\tau = n$, any two strings will bind when they

Table 1. Sizes of shuffle codes with stringency pre-assigned at 100% (first column), 50% (second column), and with the additional constraint of 50% GC content (third column), respectively

Length (n)	$\tau = n$	$\tau = \frac{n}{2}$	$\tau = \frac{n}{2} \& w = \frac{n}{2}$
12	4,096	46,656	4,096
16	65,536	1,679,620	65,536
20	1,048,580	60,466,200	1,048,580
24	16,777,200	2,176,780,000	16,777,200
28	268,435,000	78,364,200,000	268,435,000

encounter each other. The problem of information capacity of the space of DNA strands of a fixed length n for a given set of reaction conditions τ is thus reduced to the problem of finding the largest size of set of strings such that the distance between any pair of codewords is at least τ

Under these assumptions, our results can be summarized as follows. After some definitions in Section 2, a lower bound on maximal codesets with h-distance is established in Section 3 that is analogous to the the Gilbert-Varshamov sphere-packing bounds for the Hamming distance. In the next two Sections 4 and 5 we propose a novel and scalable method to construct explicitly *shuffle codes* with and without the requirement of having a constant GC-content. Their sizes do exceed the lower bound established in Section 3 and are indeed very close to the optimal in size when the stringency τ is not too large. Table 1 gives a sample for various practical lengths n and typical stringency of 100% and 50%. The last column shows the sizes of such a codeset the GC-content of whose words is constrained to 50%. Finally, as an example application, we show how these bounds and construction provide an estimate of the capacity of DNA arrays to store information on DNA chips [11], including the sensitivity of this type of representation and its range of maximal uncertainty in Section 6.

2 Gibbs Energy and the *H*-Distance

The ideal model of the hybridization process between strand pairs is the Gibbs energy released in the process [7, 18, 19], in particular for measuring the quality for a code set of strands. Although hybridization reactions *in vitro* are governed by well characterized rules of local interaction between base pairings theoretically [18] and practically [7], by models such as the nearest-neighbor model and the staggered-zipper model, difficulties arise in trying to use these rules to handle the combinatorial explosion implied in the search of good codeword *sets* in a variety of conditions, even for the small size of oligo-nucleotides of the type currently used in DNA computing (less than 150-mers.) Hence, an exhaustive search of strand sets of words maximally separated in a given coding space (say 20−mers) is infeasible because of the massive double exponential number of combinatorial possibilities that need to be checked, in principle.

To address this problem, models and methods have been considered in the literature, including various hybridization metrics [3, 13, 15, 16]. In the h-distance model, the Gibbs energy is modeled by a simple combinatorial count of the maximum number of basepair matches in the optimal alignment of two DNA strands x, y of a given length n (written from the 5′- to the 3′-end), given by

$$h(x,y) := \min_{-(n-1)<k<n-1} \left\{|k| + H(x, \sigma^k(y^{wc}))\right\} \tag{1}$$

where σ^k is the (right-) left-shift by k positions (if $k < 0$, respectively.), y^{wc} is the Watson-Crick-complement of y obtained by reversing y and exchanging A-T's, C-G's and vice versa, and $H(*,*)$ is the ordinary Hamming distance. The h-distance considers hybridization in all possible frame-shifts, which is more realistically restrictive than simpler models [15, 16] in which hybridization is considered only in the perfect alignment (i.e., shift $k = 0$). Measure 0 indicates perfect complementarity. A large measure indicates that even when x finds itself in the proximity of y, they contain few complementary basepairs, and are less likely to hybridize. This measure h can be precisely related to the maximum number of complementary base pairs in all frame shifts as follows:

Lemma 1.
$$h(x,y) = n - \max_{-(n-1)\leq i\leq n-1} \{m(x, \sigma^i(y^{wc}))\}$$

where $m(x, \sigma^i(y^{wc}))$ is the number of matches of x and y^{wc} in frame shift i.

Proof. Let s be a shift in which $h(x,y) = \tau = |s| + H(u, \sigma^s(v^{wc}))$ is minimum. In the optimal shift s, only $n - |s|$ characters are aligned. Of these, complementary basepairs are accounted for by $m(u, \sigma^s(v^{wc}))$, and uncomplementary basepairs are accounted for by $H(u, \sigma^s(v^{wc}))$. Therefore, $|s| + H(u, \sigma^s(v^{wc})) + m(u, \sigma^s(v^{wc})) = n$, and consequently, $h(x,y) = n - m(u, \sigma^s(v^{wc}))$. □

In reality, hybridization events depend on many other reactions conditions such as temperature, salinity, and kinetic factors. In the h-distance model, these parameters are all are reduced to a single numerical threshold τ. Hybridization occurs if and only if their h-distance does not exceed τ. The h-distance model is thus not kinetically accurate because it doesn't explicitly capture effects such as dissociation and re-association, the flexibility of semi-crystal DNA strands in forming loops and bulges, and the energetic contributions of mismatched pairs for example, which are explicitly excluded. It does account however, although indirectly, for the effect of temperature, salinity and other conditions in the parameter τ. The virtues of the model is therefore to be gauged, not by the physical realism with which it captures the hybridization process, but by the accuracy and feasibility of the understanding and the predictions it affords. There is good evidence that this admittedly coarse model may provide so. In recent application, it has produced results consistent with experiments *in vitro* of actual hybridizations in the range of 20-to 60-mers in various situations, such as Adleman's original solution to the Hamiltonian path problem *in silico* [11], PCR reactions [10, 12, 13]. As shown in this paper, it also helps to provide a handle on the combinatorial explosion in the search of good codewords sets for DNA-based computing and abiotic information encoding on DNA in general. (There is a technical difficulty that this measure does not satisfy all the customary properties of a mathematical metric, but can be easily be circumvented below, so we will maintain the term h-distance.)

3 Bounds on DNA Codes

To achieve efficient and scalable DNA-based computation, the set of DNA strands involved should exhibit as few cross-hybridized pairs as possible other than the planned hybridizations between complementary segments [9]. Using the h-distance model, the problem of information capacity of the space of DNA strands of a fixed length n for a given set of reaction conditions τ is thus reduced to finding the largest size of set of strings such that the h-distance between any pair of codewords is at least a given threshold τ. This number is intrinsically dependent on the "geometric" structure of the metric space of all $n-$mer, which we proceed to analyze. The problem is altogether formally analogous to the classical problem in information theory [17], except that the nature of the h-distance is essentially different from the Hamming metric and so requires new analyses and tools.

A set of DNA strands S is called an (n, τ)-code if $\forall x, y \in S$, $h(x, y) \geq \tau$. Before attempting to construct actual codes, we first provide lower and upper bounds on the possible size of a maximal (n, τ)-code.

Theorem 1. *Let $G_{n,\tau}$ to be the graph with 4^n vertices (representing all possible strands of length n) and an edge between strands u and v if $h(u, v) < \tau$. The maximum size of an (n, τ)-code S satisfies*

$$\frac{V}{1 + \frac{2E}{V}} \leq |S| \leq V$$

where $V = 4^n$ and E is the number of edges of $G_{n,\tau}$. In particular, $|S| \geq \frac{4^{n-\tau+1}}{\tau\binom{n}{\tau-1}}$.

Proof. First, it is easy to show that an independent set of $G_{n,\tau}$ is an (n, τ)-code. Applying Jensen's inequality, $\frac{\sum_{i=1}^{n} f(x_i)}{n} \geq f(\frac{\sum_{i=1}^{n} x_i}{n})$, to the convex function $f(x) = \frac{1}{x}$ with $x_i = 1 + d_i$, where d_i is the degree of vertex i, gives $\sum_{i=1}^{V} \frac{1}{1+d_i} \geq \frac{V^2}{V+2E}$. Caro's theorem [2] provides the lower bound on the size of a maximal independent set, $\alpha(G_{n,\tau})$: $|S| = \alpha(G_{n,\tau}) \geq \sum_{i=1}^{V} \frac{1}{1+d_i} \geq \frac{V}{1+\frac{2E}{V}}$.

To quantify this lower bound, we estimate the number of edges in $G_{n,\tau}$. Let H_{uv} be the event that two strands u and v hybridize. Define $H_{uv} = 1$ if $h(u, v) < \tau$, and $H_{uv} = 0$ otherwise. Then, $P(H_{uv} = 1) = P(h(u, v) \leq \tau - 1) \leq \tau\binom{n}{n-\tau+1}p^{n-\tau+1} = \tau\binom{n}{\tau-1}p^{n-\tau+1}$ The expected degree of a vertex u is $E[\frac{2E}{V}] = E[d_u] \leq \sum_v P(H_{uv}) = 4^n \cdot \tau\binom{n}{\tau-1}p^{n-\tau+1} = \tau \cdot 4^{\tau-1}\binom{n}{\tau-1}$. Now, we can apply the lower-bound result and get $|S| = \alpha(G_{n,\tau}) \geq \frac{4^n}{1+\frac{2E}{4^n}} \geq \frac{4^n}{1+\tau 4^{\tau-1}\binom{n}{\tau-1}} \approx \frac{4^{n-\tau+1}}{\tau\binom{n}{\tau-1}}$. □

This result is analogous to the Sphere-packing bound and Gilbert-Varshamov bound for Hamming codes. First note that, because the h-distance is not a true metric function, it creates a non-uniformity that makes the *volumes* of codeword spheres depend heavily on their *centers*. We can, however, consider a metric on the graph $G_{n,\tau}$ (defined in Theorem 1), by defining $d(u,v)$ to be the length of the shortest path between $u, v \in G_{n,\tau}$. Since three conditions $d(x,x) = 0, d(x,y) = d(y,x)$, and $d(x,y) + d(y,z) \geq d(x,z)$ are satisfied for all x, y, z, $(G_{n,\tau}, d)$ is indeed a metric space.

In the metric space $(G_{n,\tau}, d)$, the *expected* volume of a ball with radius 1 is $1 + \frac{2E}{V}$. In other words, the number of vertices (strands) of distance 1 away from a vertex is 1 (vertex itself) plus the average degree of $G_{n,\tau}$, which is $\frac{2E}{V}$. Similarly, the volume of a ball with radius 0 is 1 (containing the vertex itself). Since an (n,τ)-code is a maximal independent set of $G_{n,\tau}$, it can be viewed as an $(n,2)$-code in the space $(G_{n,\tau}, d)$, denoted $A(2)$. Therefore, our result means that $\frac{V}{Vol(1)} \leq |A(2)| \leq \frac{V}{Vol(0)}$, which is a probabilistic analog of the Sphere-packing Gilbert-Varshamov bounds for a Hamming code: $\frac{V}{Vol(d-1)} \leq |A(d)| \leq \frac{V}{Vol(\lfloor \frac{d-1}{2} \rfloor)}$, where V is the number of all possible codewords, and $Vol(d)$ is the number of all codewords within distance d from any strand.

4 Construction of Shuffle Codes

We now describe an efficient construction of shuffle codes and the cardinality of codes it produces, both in absolute terms and relative comparison to optimality. This construction yields a code larger code than the lower bound established in Section 3. An explicitly construction has practical significance for actual applications in larger biomolecular computing experiments and for the design of associative memories as discussed in Section 6 below. Briefly, our method refines the construction in [10] and constructs a (n,τ)-code by *shuffling* a $(\frac{n}{\tau}, 1)$-code exactly τ times. This construction is analogous to the construction of the well known Hamming codes in information theory, but with shuffling taking the place of the cyclic permutation used by Hamming [17]. Given $\{x_1, x_2, \cdots, x_k\}$, a set of DNA strands of length m, define:

$$x_1 \oslash \cdots \oslash x_k = x_{11} \cdots x_{k1} x_{12} \cdots x_{k2} \cdots x_{1m} \cdots x_{km}$$

where x_{1j} is the j^{th} character of the sequence x_1, and so on. For example, for $a_i, b_i, c_i, d_i \in \{A, C, G, T\}, 1 \leq i \leq 3$, $a_1a_2a_3 \oslash b_1b_2b_3 \oslash c_1c_2c_3 \oslash d_1d_2d_3$ results in $a_1b_1c_1d_1a_2b_2c_2d_2a_3b_3c_3d_3$. Algorithm 1 describes the construction of shuffle codes.

Theorem 2. *If $n = \tau m$, then Algorithm 1 produces an (n,τ)-code of size*

$$\left(\frac{4^m - |P_m|}{2}\right)^\tau$$

where $|P_m| = |\{x : h(x,x) = 0, |x| = m\} = 4^{\frac{m}{2}}$ if m is even, and 0 if m is odd.

Algorithm 1 Input: $n = \tau m$. Output: an (n, τ)-code in the h-distance model

```
Construct $S_m$, the set of all possible DNA strands of length $m$.
for all strands $x \in S_m$ do
  Construct $x^{wc}$, the unique complementary of $x$, i.e. $h(x, x^{wc}) = 0$.
  $S_m = S_m - x^{wc}$
end for{note: $|S_m| = \frac{4^m - |P_m|}{2}$}
Let $S = \{x_1 \oslash x_2 \oslash \cdots \oslash x_\tau : \forall x_i \in S_m\}$
Return $S$.
```

Proof. First, we want to show that $|S_m| = \frac{4^m - |P_m|}{2}$, where $|P_m| = |\{x : h(x, x) = 0, |x| = m\} = 4^{\frac{m}{2}}\}$ if m is even, and 0 if m is odd. To see that, notice that an $(m, 1)$-code contains no strands in P_m (palindromes.) Removing the complements of nonpalindromic strands those from $4^m - P_m$ (which are also nonpalindromic and at distance 0 form their matching pairs), gives us an $(m, 1)$-code of size $\frac{4^m - |P_m|}{2}$. Second, the set S produced by Algorithm 1 has $\left(\frac{4^m - |P_m|}{2}\right)^\tau$ strands, because the shuffling of each different *sets* $\{x_1, x_2, \cdots, x_\tau\}$, $x_i \in S_m$ produces a different strand $x \in S$. Further, we claim that S is an (n, τ)-code; in other words, for any two sets $\{x_1, x_2, \cdots, x_\tau\}, \{y_1, y_2, \cdots, y_\tau\}$, $x_i, y_i \in S_m$, $h(x_1 \oslash \cdots \oslash x_\tau, y_1 \oslash \cdots \oslash y_\tau) \geq \tau$. This is because in any shift of the two shuffled strands, each x_i is perfectly aligned to some y_j^{wc}. For example, consider the alignment of $a_1a_2a_3 \oslash b_1b_2b_3 \oslash c_1c_2c_3$ to $x_1x_2x_3 \oslash y_1y_2y_3 \oslash z_1z_2z_3$. In any shift, strand a can be considered *separately aligned* to either strand x, or y, or z. The same holds for strands b and c. Therefore, $h(x_1 \oslash \cdots \oslash x_\tau, y_1 \oslash \cdots \oslash y_\tau) \geq \tau \cdot h(x_i, y_j) \geq \tau \cdot 1 = \tau$. □

As a corollary, the algorithm produces a maximal (n, n)-code with $(\frac{4^1 - |P_1|}{2})^n = 2^n$. In general, shuffle codes come within a factor of $\frac{1}{2^{\tau-1}}$ from optimality.

Theorem 3. *Let L_τ be a maximal (n, τ)-code and S be the (n, τ)-code constructed by Algorithm 1. If $n = \tau m$ and if n or m is odd, then*

$$2^{\tau-1} \cdot |S| \geq |L_\tau|$$

Proof. Let L_1 be the maximal $(n, 1)$-code. Then $L_1 \geq L_\tau$, and thus $\frac{|S|}{|L_1|} \leq \frac{|S|}{|L_\tau|}$. Therefore,

$$\frac{1}{2^{\tau-1}} \leq \frac{(4^m - |P_m|)^\tau}{2^\tau} \div \frac{4^n - |P_n|}{2} = \frac{|S|}{|L_1|} \leq \frac{|S|}{|L_\tau|}$$

which yields the result. Note that the first inequality holds because $1 \leq \frac{4^n - |P_m|^\tau}{4^n - |P_n|} \leq \frac{(4^m - |P_m|)^\tau}{4^n - |P_n|}$ when n is odd or m is odd. □

Generating and storing $|S|$ strands of length n must take at least $\Omega(n \cdot |S|)$ steps and require $\Omega(n \cdot |S|)$ memory. Our algorithm is thus optimal for the construction.

Theorem 4. *Algorithm 1 takes $\Theta(n \cdot |S|)$ steps and uses $\Theta(n \cdot |S|)$ memory, where S is the (n, τ)-code returned by the algorithm.*

Proof. We assume that $n = \tau m$ for simplicity. Constructing S_m takes takes $m \cdot 4^m$ steps. This involves going through all 4^m sequences, each of which is excluded (in a $O(m)$-step check) if its reverse complement is lexicographically equal or precedes it. Constructing $S = S_m^\tau$ requires $|S| = |S_m|^\tau$ interleavings, each of which takes $O(m \cdot \tau) = O(n)$ steps. Therefore, the total number of steps is $O(m \cdot 4^m + n \cdot |S|) = O(n \cdot |S|)$, since $m \cdot 4^m \leq n \cdot |S|, \forall n > 1$. The amount of memory is never used to hold more than $O(|S|)$ strands, each of length n.

5 Shuffle Codes with Constant GC Contents

It is desirable in practice to impose an additional requirement on codeword sets that DNA strands roughly have the same amount of G's and C's [3], due to the fact the number of GC bonds (which are stronger than AT bonds) is a good measure of the melting characteristics of DNA strands. Thus, DNA strands with similar amounts of GC's have similar melting characteristics and would serve better biomolecular experiments that rely heavily on repeated melting and annealing DNA strands, such as PCR selection [9]. Using the less realistic DNA-Hamming codes, King [15] established upper and lower bounds for code sizes with constant GC content. These results do not readily carry over to the space defined by our h-distance, except in the case when $\tau = 1$. We define an (n, τ, w)-code to be an (n, τ)-code with an additional requirement that every strand must have exactly w C's and G's in total.

Algorithm 2 Input: τ, m, w. Output: an $(\tau m, \tau, \tau w)$-code in the h-distance model

```
Construct $S_m$, the set of all $m$-strands, each having exactly $w$ G's and C's in total.
for all strand $x \in S_{m,w}$ do
    Construct $x^c$, the unique complementary of $x$, i.e. $h(x, x^c) = 0$.
    $S_{m,w} = S_{m,w} - x^c$
end for{note: $|S_m| = \frac{\binom{n}{w}2^n - |Q_m|}{2}$}
Let $S = \{x_1 \oslash x_2 \oslash \cdots \oslash x_\tau : \forall x_i \in S_{m,w}$ and $x_i \neq x_j$ if $i \neq j\}$
return $S$.
```

Theorem 5. *The shuffling construction specified in Algorithm 2 can be used to construct an $(\tau m, \tau, \tau)$-code of size $m^\tau 2^{(m-1)\tau}$. Generally, Algorithm 2 can be used to construct an $(\tau m, \tau, \tau w)$-code of size*

$$\left(\frac{\binom{m}{w}2^m - |Q_m|}{2}\right)^\tau$$

where $|Q_m| = \binom{m/2}{w/2}2^{m/2}$ if n and w are even and $|Q_m| = 0$ if m or w is odd.

Proof. The set S_m constructed by Algorithm 2 is an $(m, 1, w)$-code of size $\frac{\binom{m}{w}2^m - |Q_m|}{2}$, where $Q_m = \{x : h(x, x) = 0,\ x$ has exactly w G's or C's in total.$\}$, and $|Q_m| = \binom{m/2}{w/2}2^{m/2}$ if m and w are even, and 0 if either m or w is odd. This is because there are $\binom{n}{w}$ configurations for G's and C's, in each of which there are exactly two choices for each position. We only need to discount the strands at h-distance 0. As before, these include the Watson-Crick palindromes (Q_m) and the unique complementary strands of all the strands left. Further, the number of Watson-Crick palindromes is exactly $\binom{m/2}{w/2}2^{m/2}$ because the i^{th} and $(m-i+1)^{th}$ characters of these strands must be complementary, which implies m and w must be both even and further the GC-content of w implies that each position has exactly 2 choices. Finally, as in the proof of theorem 2, the shuffling construction scales linearly both stringency and GC-content. In other words, a shuffle of τ $(m, 1, w)$-codes yields an $(\tau m, \tau, \tau)$-code. □

Theorem 6. *L_τ be a maximal $(\tau m, \tau, \tau w)$-code, and S be the $(\tau m, \tau, \tau w)$-code constructed as above. If τm or τw is odd, then*

$$2^{\tau-1} \cdot |S| \geq |L_\tau|$$

Proof. L_1, the maximal $(\tau m, 1, \tau w)$-code, is larger than L_n, and thus $\frac{|S|}{|L_1|} \leq \frac{|S|}{|L_\tau|}$. Using Sterling's approximation to factorials ($\binom{m}{w}^\tau \approx \binom{\tau m}{\tau w}$) we get

$$\frac{\binom{m}{w}^\tau}{2^{\tau-1}\binom{\tau m}{\tau w}} \leq \frac{(\binom{m}{\tau}2^m)^\tau}{2^\tau} \div \frac{\binom{\tau m}{\tau w}2^{\tau m}}{2} = \frac{|S|}{|L_1|} \leq \frac{|S|}{|L_\tau|}. \square$$

Theorem 7. *Algorithm 2 takes $\Theta(n \cdot |S|)$ steps and uses $\Theta(n \cdot |S|)$ memory, where S is the $(\tau m, \tau, \tau w)$-code returned by the algorithm.*

Proof. Similar to the proof of Theorem 4.

6 Capacity of DNA-Based Associative Memory

Garzon et al [11] propose to use DNA arrays as a medium to address the more general aspect of the encoding problem how to store abiotic information on DNA chips. In this model of associative memory, datum X is represented as a hybridization pattern given by shredding X into short fragments and pouring them over a DNA chip containing as probes a large non-crosshybridizing set S, under reaction conditions of a given stringency τ. The memory of each input is defined as the set of subsequences hybridized by the strands on the chip, counting multiplicity. Therefore, the less crosshybridizing the probe strands are on the chip, the less ambiguous their hybridization is to memory inputs, and the more inputs are unambiguously representable on the same chip. The basis elements on the chip can be obtained, for example, using Algorithm 1 or 2.

The maximum number of patterns that can be stored on a given chip and reliably retrieved unambiguously from others can be considered as a measure of the chip's capacity. An estimate of the this capacity can be obtained by estimating the probability of two inputs being indistinguishable in this representation. Let $S = \{s_1, s_2, \cdots, s_k\}$ be the basis strands on the chip. Given an input X to be stored in memory, define X_S to be the *image* of X on S as $X_S = (x_1, x_2, \cdots, x_k)$, where x_i is the number of times s_i hybridizing to (different parts) of X. Inputs X and Y are indistinguishable in S if $X_S = Y_S$, i.e. $x_i = y_i$, for all $1 \le i \le k$.

$$P(X_S = Y_S | X \neq Y) = \frac{\binom{\sigma}{x_1, x_2, \cdots, x_k}}{k^\sigma} \le \frac{2^{\sigma H(P)}}{k^\sigma} = \frac{1}{2^{\sigma(\log_2 k - H(P))}}$$

where $\sum_{i=1}^k x_i = \sum_{i=1}^k y_i = \sigma$, and $H(P) = -\frac{x_i}{\sigma} \sum_{i=1}^k \frac{x_i}{\sigma}$, the Shannon entropy of the distribution of S in X (and Y). Thus, using S as a memory is not effective if the distribution of the bases *in the inputs* is random (i.e. $H(P) \to \log_2 k$). Therefore, this type of an associative DNA memory is most effective when we can design large and noncrosshybridizing bases in such a way that their distribution in the stored inputs is as far from uniformity as possible.

7 Conclusions

We have established nearly optimal bounds on the capacity of DNA to encode and retrieve information using a combinatorial model of hybridization, the h-distance. The bounds decrease exponentially with a parameter τ that roughly codes for stringency in reaction conditions. Furthermore, we have provided a construction to produce families of error-preventive *shuffle codes*, based on a refinement of the idea in [10], that come within a constant factor of the optimal size depending on the given reaction condition parameter τ.

These results have a number of consequences for the ultimate capacity on the size of associative memories built on DNA, specifically, the capacity of DNA oligonucleotides to store abiotic information in DNA arrays as defined in [11]. According to the h-distance model, associative DNA memory is most effective if we used a large and non-crosshybridizing basis. Further, the ability of DNA memories to discriminate information is best when the distribution of stored inputs with respect to probes on the chip is as far from uniformity as possible.

Finally, there remains the issue of how realistic these results are when carried over to wet tubes. The h-distance has been used as a hybridization criterion in extensive simulations of various protocols such as Adleman's original experiment [11] and the PCR selection protocol to solve the codeword design problem [6]. The results show good agreement with the results of analogous experiments *in vitro*, both qualitatively and, within scaling limitations, quantitatively. Refinements of the model to address dissociation, re-association, and more quantitative expressions for the Gibbs energy are certainly possible. It appears reasonable to conclude at this point, however, that despite how coarse an abstraction the h-

distance may appear to be in capturing the hybridization process and reaction conditions such as temperature and salinity, the results in this paper are likely to be fairly close to the corresponding situation for DNA oligonucleotides *in vitro.*

References

1. L. Adleman (1994), Molecular computation of solutions of combinatorial problems. Science 266, 1021-1024.
2. N Alon, J. H. Spencer (2000), The Probabilistic Method, 2nd Edition, John Wiley & Sons Publisher, p. 91.
3. M. Arita, S. Kobayashi (2002), DNA Sequence Design Using Templates. New Generation Computing 20:3, 263-277. See also [14], 205-214.
4. E. Baum (1995), Building an Associative Memory Vastly larger than the Brain. Science 268, 583-585.
5. B. Brenneman, A. Condon (2001), Sequence Design for Biomolecular Computation. In press. http://www.cs.ubc.edu/~condon/papers/wordsurvey.ps.
6. R.J. Deaton, J.Chen, H. Bi, M. Garzon , H.Rubin, D.H. Wood (2002), A PCR-based protocol for In Vitro Selection of Non-crosshybridizing Oligonucleotides. In: [14], 105-114.
7. R.J. Deaton, J. Chen, H. Bi, J.A. Rose (2002b), A Software Tool for Generating Non-crosshybridizing Libraries of DNA Oligonucleotides. In: [14], 211-220.
8. R. Deaton, M. Garzon, R. E. Murphy, J. A. Rose, D. R. Franceschetti, S.E. Stevens, Jr. (1998), The Reliability and Efficiency of a DNA Computation. Phys. Rev. Lett. 80, 417.
9. M.H. Garzon, R.J. Deaton (2004), Codeword Design and Information Encoding in DNA Ensembles. J. of Natural Computing, in press.
10. M.H. Garzon, K. Bobba, B.P Hyde. (2004), Digital Information Encoding on DNA. Aspects of Molecular Computing 2004: 152-166.
11. M. Garzon, D. Blain, K. Bobba, A. Neel, M. West. (2003), Self-Assembly of DNA-like structures In-Silico. Genetic Programming and Evolvable Machines 4:2, 185-200.
12. M. Garzon, P.I. Neathery, R. Deaton, R.C. Murphy, D.R. Franceschetti, S.E. Stevens, Jr. (1997), A New Metric for DNA Computing. Proc. 2nd Annual Genetic Programming Conference. Morgan Kaufmann, 472-478.
13. M. Garzon, R. Deaton, P. Neathery, R.C. Murphy, D.R. Franceschetti, E. Stevens Jr. (1997), On the Encoding Problem for DNA Computing. Poster at The Third DIMACS Workshop on DNA-based Computing, U of Pennsylvania. Preliminary Proceedings, 230-237.
14. M. Hagiya, A. Ohuchi, eds. (2002), Proc. 8th Int. Meeting on DNA-Based Computers. Springer-Verlag Lecture Notes in Computer Science LNCS 2568. Springer-Verlag.
15. O. King (2003), Bounds for DNA codes with constant GC-content, Journal of Combinatorics, 10(1) R33 13pp, 2003.
16. A. Marathe, A. Condon, R. Corn (1999), On Combinatorial DNA Word Design. Proceedings 5th DIMACS Workshop on DNA Based Computers. American Mathematical Society. Editors: Erik Winfree and David K. Gifford. 75-89.
17. J. Roman (1995), The Theory of Error-Correcting Codes. Springer-Verlag, Berlin.

18. J. SantaLucia, Jr., H.T. Allawi, P.A. Seneviratne (1990), Improved Nearest Neighbor Paramemeters for Predicting Duplex Stability. Biochemistry 35, 3555-3562.
19. J.G. Wetmur (1997), Physical Chemistry of Nucleic Acid Hybridization. American Mathematical Society DIMACS Series 48 (1999). 1- 23.

Compact Error-Resilient Computational DNA Tiling Assemblies*

John H. Reif, Sudheer Sahu, and Peng Yin

Department of Computer Science, Duke University, Box 90129,
Durham, NC 27708-0129, USA
{reif, sudheer, py}@cs.duke.edu

Abstract. The self-assembly process for bottom-up construction of nanostructures is of key importance to the emerging scientific discipline Nanoscience. However, self-assembly at the molecular scale is prone to a quite high rate of error. Such high error rate is a major barrier to large-scale experimental implementation of DNA tiling. The goals of this paper are to develop theoretical methods for compact error-resilient self-assembly and to analyze these methods by stochastic analysis and computer simulation. Prior work by Winfree provided an innovative approach to decrease tiling self-assembly errors without decreasing the intrinsic error rate ϵ of assembling a single tile. However, his technique resulted in a final structure that is four times the size of the original one. This paper describes various *compact* error-resilient tiling methods that *do not increase the size of the tiling assembly*. These methods apply to assembly of boolean arrays which perform input sensitive computations (among other computations). Our 2-way (3-way) overlay redundancy construction drops the error rate from ϵ to approximately ϵ^2 (ϵ^3), without increasing the size of the assembly. These results were further validated using stochastic analysis and computer simulation.

1 Introduction

Self-assembly is a process in which simple objects associate into large (and complex) structures. The self-assembly of DNA tiles can be used both as a powerful computational mechanism [4, 6, 10, 11, 14] and as a bottom-up nanofabrication technique [8]. Periodic 2D DNA lattices have been successfully constructed with a variety of DNA tiles, for example, double-crossover (DX) DNA tiles [13], rhombus tiles [5], triple-crossover (TX) tiles [3], and 4x4 tiles [15]. Two dimensional algorithmic self-assembly, in contrast, is comparatively resistant to experimental demonstration, partially due to the large number of errors in the assembled structure.

How to decrease such errors? There are primarily two kinds of approaches. The first one is to decrease the intrinsic error rate ϵ by optimizing the physical environment in which a fixed tile set assembles [11], by improving the design of the tile itself using new molecular mechanism [2], or by using novel materials. The second approach is to design new tile sets that can reduce the total number of errors in the final structure even

* Extended abstract. For full paper, see [7]

with the same intrinsic error rate. A seminal work in this direction is the proofreading tile set constructed by Winfree [12].

One desirable improvement on Winfree's construction (which results in an assembled structure with 4x size of the original one) is to make the design more compact. Here we report construction schemes that achieve performance similar as or better than Winfree's tile set without scaling up the assembled structure. We will describe our work primarily in the context of self-assembling Sierpinsky triangles and binary counters, but note that the design principle can be applied to a more general setting. The basic idea of our construction is to overlay redundant computations and hence force consistency in the scheme. The idea of using redundancy to enhance reliability of a system constructed from unreliable individual components goes back to von Neumann [9].

The rest of the paper is organized as follows. In Sect. 2, we introduce the assembly problem. In Sect. 3, we describe a scheme that decreases the error rate from ϵ to $6\epsilon^2$. In Sect. 4, this scheme is further improved to $30\epsilon^3$ using a three-way overlay redundancy technique. Kinetic analysis is performed in Sect. 5 to show that the assembly speed is not much decreased. Sect. 6 gives empirical study using computer simulation. We conclude with discussions about future work in Sect. 7.

2 Assembly with No Error Corrections

2.1 Assembly Problems

A general assembly problem considered in this paper is the assembly of *a Boolean array*. A Boolean array assembly is an $N \times M$ array, where the elements of each row are indexed over $\{0, \ldots, N-1\}$ from right to left and the elements of each column are indexed over $\{0, \ldots, M-1\}$ from bottom to top. The bottom row and right most column both have some given values. Let $V(i,j)$ be the value of the i-th (from the right) bit on the j-th row (from the bottom) displayed at position (i,j) and communicated to the position $(i, j+1)$. Let $U(i,j)$ be a Boolean value communicated to the position $(i+1, j)$. For $i = 1, \ldots, N-1$ and $j = 1, \ldots, M-1$, we have $V(i,j) = U(i-1,j)\ OP_1\ V(i,j-1)$ and $U(i,j) = U(i-1,j)\ OP_2\ V(i,j-1)$, where OP_1 and OP_2 are two Boolean functions, each with two Boolean arguments and one Boolean output. See Figure 1 for an illustration.

Two Boolean arrays of particular interest are the *Sierpinsky Triangle* [1] and the *Binary counter*. The Sierpinsky Triangle is an $N \times N$ Boolean binary array, where the bottom row and right most column all have 1s; its OP_1 and OP_2 operators are both XOR. Recall that XOR is exclusive OR, a binary operator that outputs bit 1 if the two input bits are different and 0 otherwise. For an illustration, see Figure 2.

The binary counter is an $N \times 2^N$ Boolean binary array. In a binary counter, the bottom row has all 0s and the j-th row (from the bottom) is the binary representation of counter value j, for $j = 0, \ldots, 2^N - 1$. Note that the i-th bit is i-th from the right – this is in accordance with the usual left to right binary notation of lowest precision bits to highest precision bits. $V(i,j)$ represents the value of the i-th (from the right) counter bit on the j-th row (from the bottom), and $U(i,j)$ is the value of the carry bit from the counter bit at position (i,j). In the binary counter, we have $V(0,j) = V(0,j-1)\ XOR\ 1$; $V(i,j) = U(i-1,j)\ XOR\ V(i,j-1)$ for $i = 1, \ldots, N-1$;

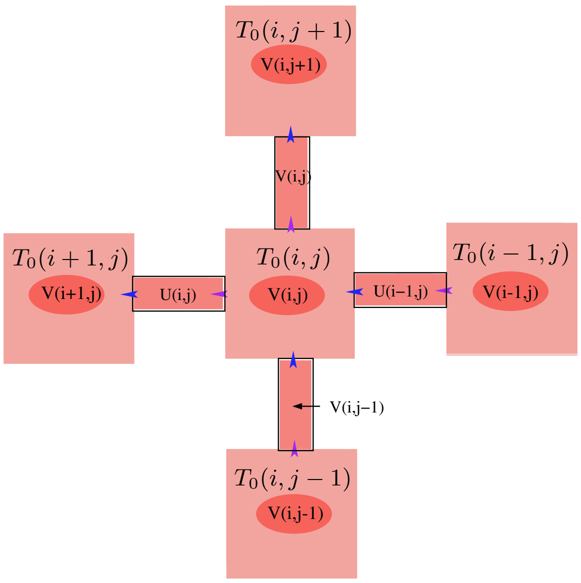


Fig. 1. Tile $T_0(i,j)$ takes input $U(i-1,j)$ and $V(i,j-1)$; determines $V(i,j) = U(i-1,j)\ OP_1\ V(i,j-1)$ and $U(i,j) = U(i-1,j)\ OP_2\ V(i,j-1)$; displays $V(i,j)$

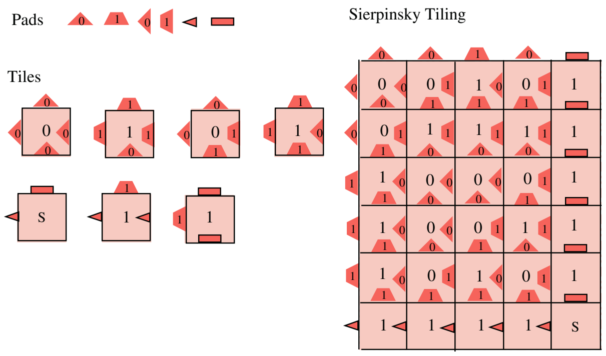


Fig. 2. Sierpinsky triangle tiling assembly

$U(i,j) = U(i-1,j)\ AND\ V(i,j-1)$. Hence OP_1 is the XOR operation and OP_2 is the AND operation.

We observe that OP_1 is XOR both for the Sierpinsky Triangle and for the binary counter and we will thus assume in our error-resilient constructions that OP_1 is XOR. Each assembly will be constructed with the 4×4 DNA tiles described in [15]. A 4×4 tile allows one pad per side (which can communicate a small constant number of bits). Furthermore, we will assume that a "frame" is assembled first for each binary array, consisting of a "bottom row" with N horizontally aligned tiles and a "right border" linear assembly with M vertically aligned tiles.

2.2 Assemblies with No Error Corrections

We first describe the naive assembly scheme without error correction. *Note that such assembly requires only 4 tile types in addition to 3 frame tiles, but results in rather small scale error-free assemblies (with the actual size contingent on the probability of single pad mismatch between adjacent tiles).* We call this scheme version 0 assembly and denote the tiles as $T_0(i, j)$.

The simplest way to construct such an assembly is to make each side of each tile a binary valued pad. Since the values of the left and top pads depend on the values of the right and bottom pads, the tile type depends on only 2 binary pads, and hence only $2^2 = 4$ tile types are required in addition to the 3 tiles for assembling the initial frame.

The bottom, right, top, and left pads of tile $T_0(i, j)$ represent the values of $V(i, j-1)$ (as communicated from the tile below $T_0(i, j-1)$), $U(i-1, j)$ (as communicated from the tile on its right $T_0(i-1, j)$), $V(i, j)$ (as computed by $V(i, j-1)\ OP_1\ U(i-1, j)$), and $U(i, j)$ (as computed by $V(i, j-1)\ OP_2\ U(i-1, j)$), respectively. A determined value $V(i, j) = 1$ can be displayed by the tile $T_0(i, j)$ using, for example, an extruding stem loop of single strand DNA.

2.3 Errors in Assemblies

All this is theoretically correct, but it has not taken into account the error rate of the assembly of individual DNA tiles. A critical issue in 2D tiling assemblies is the pad mismatch rate, which determines the size of the error-free assembly. Let ϵ be the probability of a single pad mismatch between adjacent assembling DNA tiles, and assume that the likelihood of a pad mismatch error is independent for distinct pads as long as they do not involve the binding of the same two tiles. As such, a pad mismatch rate of $\epsilon = 5\%$ would imply an error-free assembly with an expected size of only 20 tiles, which is disappointingly small. Thus, a key challenge in experimentally demonstrating large scale algorithmic assemblies is to construct error-resilient tiles. Winfree's construction is an exciting step towards this goal [12]. However, to reduce the error rate to ϵ^2 (resp. ϵ^3), his construction replaces each tile with a group of $2 \times 2 = 4$ (resp. $3 \times 3 = 9$) tiles and hence increases the size of the tiling assembly by a factor of 4 (resp. 9). Our construction described below, in contrast, reduces the tiling error rate without scaling up the size of the final assembly. This would be an attractive feature in the attempt to obtain assemblies with large computational capacity. We call our construction *compact error resilient assemblies* and describe them below in detail.

3 Error-Resilient Assembly Using Two-Way Overlay Redundancy

3.1 Construction

To achieve the goals stated in previous section, we propose the following error resilient tiling scheme. *Our Error-Resilient Assembly I (using two-way overlay redundancy) uses only 8 computational tile types plus the 4 frame tile types. This drops the probability of* *assembly error to* $6\epsilon^2$*, which is* 1.5% *for* $\epsilon = 5\%$*, potentially allowing for error-free assemblies of expected size in the hundreds of tiles.*

The construction is depicted in Figure 3. Tiles in this construction are denoted as T_1 tiles (for version 1). Each pad of each tile encodes a pair of bits. The basic idea of this Error-Resilient assembly is the *two-way overlay redundancy*: each tile $T_1(i, j)$ computes the outputs for its own position (i, j) and also for its right neighbor's position $(i - 1, j)$; the redundant computation results obtained by $T_1(i, j)$ and its right neighbor $T_1(i-1, j)$ are compared via an additional *error checking portion* on $T_1(i, j)$'s right pad (which is the same as $T_1(i - 1, j)$'s left pad). Tile $T_1(i, j)$'s right neighbor $T_1(i - 1, j)$ is not likely to bind to $T_1(i, j)$ if these pad values are not consistent. Hence if only one of $T_1(i, j)$ and $T_1(i - 1, j)$ is in error (incorrectly placed), the kinetics of the assembly may allow the incorrectly placed tile to be ejected from the assembly.

The four pads of $T_1(i, j)$ are constructed as follows (Figure 3).

- The right and left portions of the bottom pad represent the value of $V(i - 1, j - 1)$ and $V(i, j - 1)$ respectively as communicated from the tile $T_1(i, j - 1)$.
- The top portion of the right pad represents the value of $U(i-2, j)$ as communicated from the tile $T_1(i - 1, j)$. The bottom portion of the right pad represents the value of $V(i - 1, j)$ as determined by the tile $T_1(i, j)$. Note that the value $V(i - 1, j)$ is also redundantly determined by $T_1(i-1, j)$ and hence the bottom portion performs comparison of the two values and is referred to as *error checking portion*, and labeled with checked background in Figure 3.

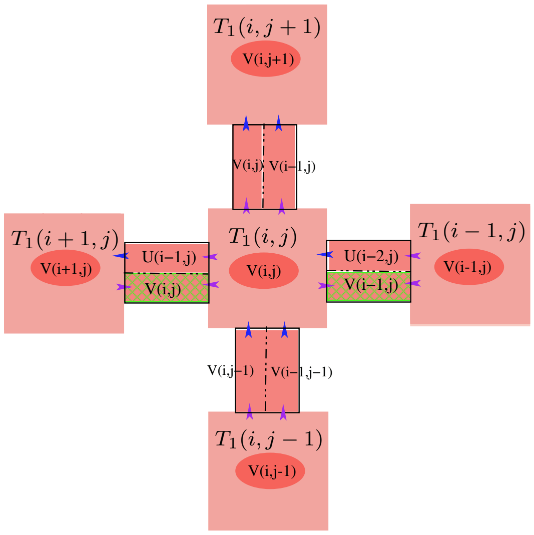


Fig. 3. Construction of compact error-resilient assembly version I. Each pad has two portions. A portion encoding an input (resp. output) value is indicated with a dark blue (resp. light pink) colored arrow head. The error checking portion is depicted as a checked rectangle. Tile $T_1(i, j)$ takes inputs $U(i - 2, j)$, $V(i - 1, j - 1)$, and $V(i, j - 1)$; determines $V(i - 1, j) = U(i - 2, j)\ OP_1\ V(i - 1, j - 1)$, $U(i - 1, j) = U(i - 2, j)\ OP_2\ V(i - 1, j - 1)$, and $V(i, j) = U(i - 1, j)\ OP_1\ V(i, j - 1)$; displays $V(i, j)$

- The top and bottom portions of the left pad represent the values of $U(i-1,j)$ and $V(i,j)$ respectively, as determined by the tile $T_1(i,j)$. Again, the bottom portion is the error checking portion.
- The right and left portions of the top pad represent the values of $V(i-1,j)$ and $V(i,j)$ respectively, as determined by tile $T_1(i,j)$.

The above tile design allows the values $V(i-1,j-1)$ and $V(i,j-1)$ to be communicated to tile $T_1(i,j)$ from the tile $T_1(i,j-1)$ just below $T_1(i,j)$. The value $U(i-2,j)$ is communicated to tile $T_1(i,j)$ from its immediate right neighbour $T_1(i-1,j)$. The values $V(i-1,j)$ and $U(i-1,j)$ are determined by tile $T_1(i,j)$ from $V(i-1,j-1)$ and $U(i-2,j)$: $V(i-1,j) = U(i-2,j)\ OP_1\ V(i-1,j-1)$ and $U(i-1,j) = U(i-2,j)\ OP_2\ V(i-1,j-1)$. The value $V(i,j)$ is determined from $V(i,j-1)$ and $U(i-1,j)$: $V(i,j) = U(i-1,j)\ OP_1\ V(i,j-1)$. If the determined value $V(i,j)=1$, then it is displayed by the tile $T_1(i,j)$.

In this construction, each pad encodes two bits. However, since the values of the left pad, the top pad, and the bottom portion ($V(i-1,j)$) of the right pad each depend only on the values of the top portion ($U(i-2,j)$) of the right pad and the bottom pads, the tile type depends on only 3 input binary bits. Hence only $2^3 = 8$ tile types are required. In addition, 4 tiles are required to assemble the frame, as described in Sect. 6.

We emphasize that though a pad has two portions, it should be treated as a whole unit. A value change in one portion of a pad changes the pad to a completely new pad. If the pad is implemented as a single strand DNA, this means that the sequence of the single strand DNA will be a complete new sequence. One potential confusion to be avoided is mistakenly considering two pads encoding, say 00 and 01, as having the 0 portions identical or, in the context of single strand DNA, as having half of the DNA sequences identical. To emphasize the unity of a pad, we put a box around each pad in Figure 3.

3.2 Error Analysis

Recall that ϵ is the probability of a single pad mismatch between two adjacent DNA tiles. We further assume that the likelihood of a pad mismatch error is independent for distinct pads as long as they do not involve the binding of the same two tiles and that OP_1 is the function XOR.

Our intention is that the individual tiling assembly error rate (and hence the propagation of these errors to further tile assemblies) is substantially decreased, due to co-operative assembly of neighboring tiles, which redundantly compute the $V(-,-)$ and $U(-,-)$ values at their positions and at their right neighbours.

Without loss of generality, we consider only the cases where the pad binding error occurs on either the bottom pad or the right pad of a tile $T_1(i,j)$. Otherwise, if the pad binding error occurs on the left (resp. top) pad of tile $T_1(i,j)$, then use the same below argument for tile $T_1(i+1,j)$ (resp. $T_1(i,j+1)$). We define the *neighborhood* of tile $T_1(i,j)$ to be the set of 8 distinct tiles $\{\ T_1(i',j') : |i'-i| < 2,\ |j'-j| < 2\ \} \setminus \{\ T_1(i,j)\ \}$ with coordinates that differ from (i,j) by at most 1. A neighborhood tile $T_1(i',j')$ is *dependent* on $T_1(i,j)$ if both its coordinates are equal to or greater than those of $T_1(i,j)$; otherwise $T_1(i',j')$ is *independent* of $T_1(i,j)$. Note that a neighborhood tile $T_1(i',j')$ is dependent on $T_1(i,j)$ if and only if the values $V(i',j')$ and

$U(i', j')$ are determined at least partially from $V(i, j)$ or $U(i, j)$. More specifically, the neighborhood tiles dependent on $T_1(i, j)$ are $T_1(i+1, j+1)$, $T_1(i+1, j)$, and $T_1(i, j+1)$. The neighborhood tiles independent of $T_1(i, j)$ are $T_1(i+1, j-1)$, $T_1(i, j-1)$, $T_1(i-1, j+1)$, $T_1(i-1, j)$, and $T_1(i-1, j-1)$.

Lemma 1. *Suppose that the neighborhood tiles independent of tile $T_1(i, j)$ have correctly computed $V(-,-)$ and $U(-,-)$. If there is a single pad mismatch between tile $T_1(i, j)$ and another tile just below $T_1(i, j)$ or to its immediate right, then there is at least one further pad mismatch in the neighborhood of tile $T_1(i, j)$. Furthermore, given the location of the initial mismatch, the location of the further pad mismatch can be determined among at most three possible pad locations.*

Proof. Suppose that a pad binding error occurs on the bottom pad or the right pad of tile $T_1(i, j)$ but no further pad mismatch occurs between two neighborhood tiles which are independent of $T_1(i, j)$. We now consider a case analysis of possible pad mismatches.

(1) First consider the case where the pad binding error occurs on the bottom pad of tile $T_1(i, j)$. Recall that the right and left portions of the bottom pad represent the values of $V(i-1, j-1)$ and $V(i, j-1)$ respectively as communicated from tile $T_1(i, j-1)$. Observe that neighborhood tiles $T_1(i, j-1)$, $T_1(i-1, j-1)$, and $T_1(i-1, j)$ are all independent of $T_1(i, j)$ and so all *correctly* compute $V(-,-)$ and $U(-,-)$ according to the assumption of the lemma.

(1.1) Consider the case where the pad binding error is due to the *incorrect* value of the right portion $V(i-1, j-1)$ of the bottom pad of tile $T_1(i, j)$ as shown in Figure 4. Note that the left portion $V(i, j-1)$ of the bottom pad of tile $T_1(i, j)$ may also

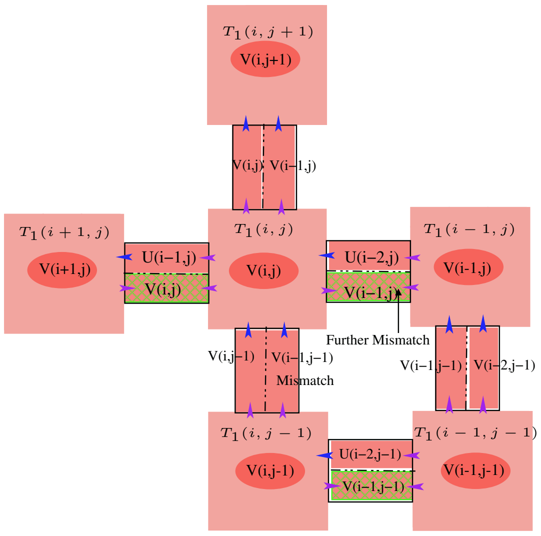


Fig. 4. Case 1.1 in the proof of Lemma 1: error in right portion $V(i-1, j-1)$ of the bottom pad of tile $T_1(i, j)$ causes a further mismatch on the right pad of tile $T_1(i, j)$

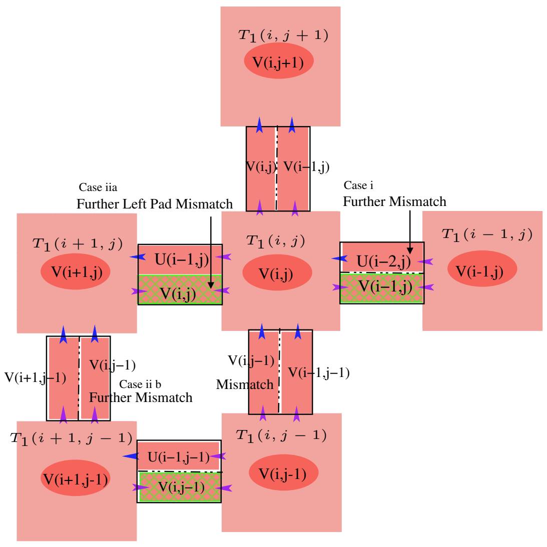


Fig. 5. Case 1.2 in the proof of Lemma 1: a further mismatch is caused by an error in the $V(i, j-1)$ portion of the bottom pad of tile $T_1(i, j)$

be *incorrect*. In case (i), $T_1(i, j)$ has an *incorrect* value for the $U(i-2, j)$ portion of its right pad and hence there is a further pad mismatch on the right pad of $T_1(i, j)$. In case (ii), $T_1(i, j)$ has a *correct* value for the $U(i-2, j)$ portion of its right pad. Since $T_1(i, j)$ uses the formula $V(i-1, j) = U(i-2, j)\ OP_1\ V(i-1, j-1)$ to compute $V(i-1, j)$ and OP_1 is assumed to be the XOR function, it will determine an *incorrect* value for $V(i-1, j)$, which is distinct from the *correct* value of $V(i-1, j)$ determined by its (independent) right neighbor tile $T_1(i-1, j)$. This again implies a further pad mismatch on the right pad of tile $T_1(i, j)$.

(1.2) Next consider the case in Figure 5 where the pad binding error is due to the wrong value of the left portion $V(i, j-1)$ of the bottom pad of tile $T_1(i, j)$. However, there is a *correct* match in the right portion $V(i-1, j-1)$ of the bottom pad of tile $T_1(i, j)$. In case (i), $T_1(i, j)$ has an *incorrect* value for the top portion $U(i-2, j)$ of its right pad, then there will be a mismatch on the right pad of $T_1(i, j)$. In case (ii), $T_1(i, j)$ has a *correct* value for the top portion $U(i-2, j)$ of its right pad, then it will further determine a *correct* value for $U(i-1, j)$, since $U(i-1, j) = U(i-2, j)\ OP_2\ V(i-1, j-1)$ and both $U(i-2, j)$ and $V(i-1, j-1)$ have correct values. Since $V(i, j) = U(i-1, j)\ OP_1\ V(i, j-1)$, $U(i-1, j)$ is correct and $V(i, j-1)$ is incorrect, $T_1(i, j)$ will determine an *incorrect* value for $V(i, j)$.

Note that the neighborhood tiles $T_1(i-1, j-1)$, $T_1(i, j-1)$, and $T_1(i+1, j-1)$ are independent of $T_1(i, j)$ and so both *correctly* compute $V(-,-)$ and $U(-,-)$. However, $T_1(i, j)$'s immediate left neighbour $T_1(i+1, j)$ is dependent both on the *incorrect* value communicated by the pad of $T_1(i, j)$ and the *correct* values communicated by the pad of $T_1(i+1, j-1)$. So in case (ii) there must be a further pad mismatch at tile $T_1(i+1, j)$ as argued below. In case (iia) there is pad mismatch on the right pad of $T_1(i+1, j)$ either due to a mismatch on the portion of $U(i-1, j)$ or on the portion of $V(i, j)$. Otherwise, in case (iib) there is no mismatch on either the $U(i-1,j)$ or the $V(i,j)$ portion of the pad between $T_1(i,j)$ and $T_1(i+1,j)$. This implies that $V(i,j)$ is *incorrectly* computed by $T_1(i+1,j)$ (since $T_1(i,j)$ has incorrectly computed $V(i,j)$), but $T_1(i+1,j)$ has a correct value of $U(i-1,j)$. However, $V(i,j) = U(i-1,j)\ OP_1\ V(i,j-1)$ and OP_1 is XOR, this implies that the right portion $V(i,j-1)$ of the bottom pad of $T_1(i+1,j)$ has an *incorrect* value, and hence there is a mismatch between $T_1(i+1,j)$ and $T_1(i+1,j-1)$.

(2) Next consider the case where the pad binding error occurs on the right pad of tile $T_1(i,j)$, but there is no error on the bottom pad of $T_1(i,j)$. We first note that the value of the top portion $U(i-2,j)$ of the right pad of $T_1(i,j)$ must have an *incorrect* value. Assume the opposite case where $U(i-2,j)$ is correct. But the $V(i-1,j-1)$ portion of $T_1(i,j)$'s bottom pad must also have a correct value (no mismatch on the bottom pad), this results in a further correct value for the $V(i-1,j)$ portion of $T_1(i,j)$'s right pad. Thus both $U(i-2,j)$ and $V(i-1,j)$ portions of $T_1(i,j)$'s right pad are correct and there must be no mismatch on the right pad. A contradiction. Therefore, $U(i-2,j)$ must have an *incorrect* value, and hence we only need to consider this case.

(2.1) Now consider the case where the pad binding error is due to the *incorrect* value of the top portion $U(i-2,j)$ of the right pad of tile $T_1(i,j)$ as shown in Figure 6. We note that $T_1(i,j)$ will compute an *incorrect* value for the right portion $V(i-1,j)$ of its top pad, according to the formula $V(i-1,j) = U(i-2,j)\ OP_1\ V(i-1,j-1)$. Note that $T_1(i,j+1)$ is dependent on $T_1(i,j)$. In case (i), tile $T_1(i,j+1)$ has a *correct* value of $V(i-1,j)$. There must be a pad mismatch on $V(i-1,j)$ between $T_1(i,j+1)$ and $T_1(i,j)$, since the value of $V(i-1,j)$ determined by $T_1(i,j)$ is incorrect. In case (ii),

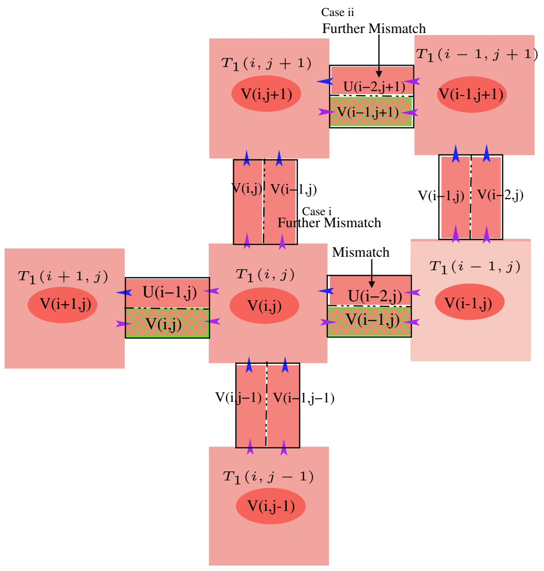


Fig. 6. Case 2.1 in the proof of Lemma 1: a further mismatch is caused by an error in the $U(i-2,j)$ portion of the right pad of tile $T_1(i,j)$

tile $T_1(i, j+1)$ has an *incorrect* value of $V(i-1, j)$, using similar argument as in case 1.1, we can show that there must be a pad mismatch on the $U(i-2, j+1)$ portion of $T_1(i, j+1)$'s right pad.

Hence we conclude that in each case, there is a further pad mismatch between a pair of adjacent tiles in the neighborhood of tile $T_1(i, j)$. Furthermore, we have shown in each case that given the location of the initial mismatch, the location of the further pad mismatch can be determined among at most three possible pad locations. □

Recall that we have let ϵ be the probability of a single pad mismatch between adjacent assembling tiles. This implies that $1-\epsilon$ is the probability of no single pad mismatch between a given pair of adjacent tiles. So the probability that there is no pad mismatch between tile $T_1(i, j)$ and another tile just below or to its immediate right is $(1-\epsilon)^2$. Hence the probability that there is a pad mismatch between tile $T_1(i, j)$ and another tile just below or to its immediate right is $1-(1-\epsilon)^2 = 2\epsilon - \epsilon^2$, which is at most 2ϵ. But by Lemma 1, if there is a pad mismatch between tile $T_1(i, j)$ and another tile just below or to its immediate right, then there is a further pad mismatch between a pair of adjacent tiles in the immediate neighborhood of tile $T_1(i, j)$, and the location of the further pad mismatch can be determined among at most three possible pad locations. The probability that there is such a further pad mismatch between tiles at most three possible pad locations is at most $1-(1-\epsilon)^3$, which is at most 3ϵ. This implies that with probability at most $(3\epsilon)(2\epsilon) = 6\epsilon^2$, there is both (i) a pad mismatch between tile $T_1(i, j)$ and another tile just below or to its immediate right; and (ii) furthermore, there is also a further pad mismatch between tiles in the immediate neighborhood of tile $T_1(i, j)$ as considered in the case analysis in the proof of Lemma 1. Hence we have shown:

Theorem 1. *Suppose that the neighborhood tiles independent of tile $T_1(i, j)$ have correctly computed $V(-,-)$ and $U(-,-)$. Then the assembly error probability for tile $T_1(i, j)$ is at most $6\epsilon^2$, where ϵ is the probability of a single pad mismatch.*

4 Error-Resilient Assembly Using Three-Way Overlay Redundancy

4.1 Construction

We next extend the design of our scheme to a 3-way overlay scheme. The Error-Resilient Assembly II (using 3-way overlay redundancy) uses 16 computational tile types and 5 frame tile types. One mismatch on a tile forces two more mismatches in its neighborhood. This property further lowers the assembly error.

The basic construction is shown in Figure 7. In this construction, each pad encodes a tuple of 3 bits and hence is an 8-valued pad. The basic idea of this error-resilient assembly is to have each tile $T_2(i, j)$ compute error checking values for positions $(i-1, j)$, $(i, j-1)$, $(i+1, j)$, and $(i, j+1)$, which are compared with corresponding error checking values computed by $T_2(i, j)$'s four neighbors. The neighbors are unlikely to bind with $T_2(i, j)$ if such error checking values are inconsistent, and the kinetics of the assembly will allow these tiles to dissociate from each other, as in version 1 (2-way overlay redundancy). However, instead of introducing just one additional mismatch in $T_2(i, j)$'s

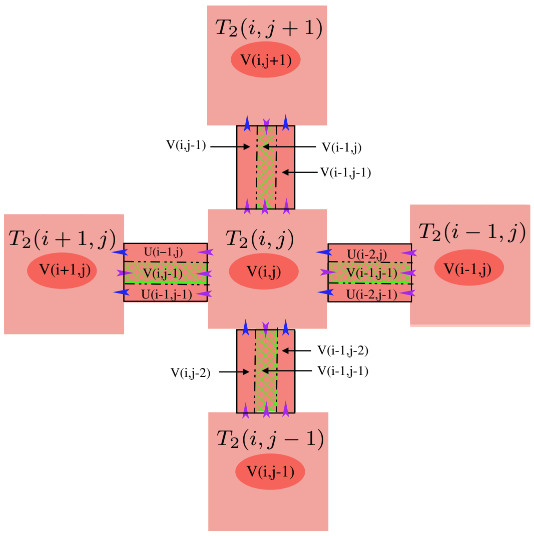


Fig. 7. Tile T_2 takes inputs $U(i-2,j)$, $U(i-2,j-1)$, $V(i-1,j-2)$, and $V(i,j-2)$; determines $V(i-1,j-1) = U(i-2,j-1)\ OP_1\ V(i-1,j-2)$, $U(i-1,j-1) = U(i-2,j-1)\ OP_2\ V(i-1,j-2)$, $V(i,j-1) = U(i-1,j-1)\ OP_1\ V(i,j-2)$, $U(i-1,j) = U(i-2,j)\ OP_2\ V(i-1,j-1)$, $V(i,j) = U(i-1,j)\ OP_1\ V(i,j-1)$, and $V(i-1,j) = U(i-2,j)\ OP_1\ V(i-1,j-1)$; displays $V(i,j)$

neighborhood, the 3-way overlay redundancy (version 2) forces two mismatches, and hence we have a further lowered error rate.

4.2 Error Analysis

For error analysis, in addition to the assumptions made in Sect. 3.2, we require that OP_2 can detect incorrect value of input 1 regardless of the correctness of input 2. This property seems essential to guarantee two further mismatches in a tile's neighborhood when there is an initial mismatch on one of the tile's four pads.

Using a similar but more involved analysis as in Lemma 1 and Theorem 1, we can show

Lemma 2. *Suppose that the neighborhood tiles independent of tile $T_2(i,j)$ have correctly computed $V(-,-)$ and $U(-,-)$. If there is a single pad mismatch between tile $T_2(i,j)$ and another tile just below or to its immediate right, then there are at least two further pad mismatches between pairs of adjacent tiles in the immediate neighborhood of tile $T_2(i,j)$. Furthermore, given the location of the initial mismatch, the location of the second mismatch can be determined among at most three locations in the neighborhood of $T_2(i,j)$; given the location of the initial and the second mismatches, the location of the third mismatch can be determined among at most five locations.*

Theorem 2. *Suppose that the neighborhood tiles independent of tile $T_2(i,j)$ have correctly computed $V(-,-)$ and $U(-,-)$. Then the assembly error probability for tile*

$T_2(i, j)$ is at most $2\epsilon \times 3\epsilon \times 5\epsilon = 30\epsilon^3$, where ϵ is the probability of a single pad mismatch.

For detailed analysis, see [7]. It is also easy to see that this schemes requires $2^4 = 16$ computational tile types and 5 frame tile types.

5 Kinetic Analysis

Our kinetic analysis is based upon the analysis done by Winfree [12]. Two parameters, G_{se} and G_{mc}, are defined in [12]. G_{mc} measures the entropic cost of fixing the location of a monomer unit and G_{se} measures the free energy cost of breaking a single sticky-end bond. A non-rigorous condition for good self-assembly is given as $G_{mc} \approx 2G_{se}$ and the growth rate of assembly r is approximately $\alpha e^{-G_{mc}}$.

For the construction with no error-correction, the equilibrium error rate δ_0 for an assembly is approximately $k_0 e^{-G_{se}}$, which yields an assembly rate $r_0 \approx \alpha e^{-G_{mc}} \approx \alpha e^{-2G_{se}} \approx (\alpha/k_0^2)\delta_0^2$ [12]. For our version 1 error-resilient construction, it can be shown that $\delta_1 \approx k_1 e^{-2G_{se}}$, which further yields $r_1 \approx \alpha e^{-G_{mc}} \approx \alpha e^{-2G_{se}} \approx (\alpha/k_1)\delta_1$. For our version 2 error resilient scheme, it can be shown that the error rate is approximately $\delta_2 \approx k_2 e^{-3G_{se}}$, which yields $r_2 \approx \alpha e^{-G_{mc}} \approx \alpha e^{-2G_{se}} \approx (\alpha/k_2)\delta_2^{(2/3)}$. k_0, k_1 and k_2 are constants. See [7] for details. The above analysis shows that while the error rates δ_1 and δ_2 are significantly reduced in our error resilient assemblies, the aggregation speeds r_1 and r_2 stay approximately the same as r_0.

6 Computer Simulation

We first give below the construction of a Sierpinsky Triangle using our error resilient assembly version 1, and then perform empirical study of the error rates using computer simulation of assembly of the Sierpinsky Triangle and compare the results with that of Winfree's [12].

Figure 8 illustrates the construction of a Sierpinsky triangle, using 8 computational tiles and 4 frame tiles. We would like to again emphasize that although we give the construction of the tiles in previous sections with each pad having two or three distinct portions, a mismatch on any portion of a pad results in a *total* mismatch of the whole pad instead of a partial mismatch of only that portion. Hence, in Figure 8, we use a distinct label for each pad, emphasizing the wholeness of the pad.

For the simulation study, we used the Xgrow simulator by Winfree [12] and simulated the assembly of Sierpinsky triangles for the following cases:

– assembly without any error correction,
– assembly using Winfree's 2×2 proofreading tile set,
– assembly using Winfree's 3×3 proofreading tile set,
– assembly using our error resilient scheme version 1, T_1 (construction in Figure 8),
– assembly using our error resilient scheme version 2, T_2 (construction not shown).

We performed simulations of the assembly process of a target aggregate of 512×512 tiles. A variable N is defined as the number of tiles assembled without any permanent

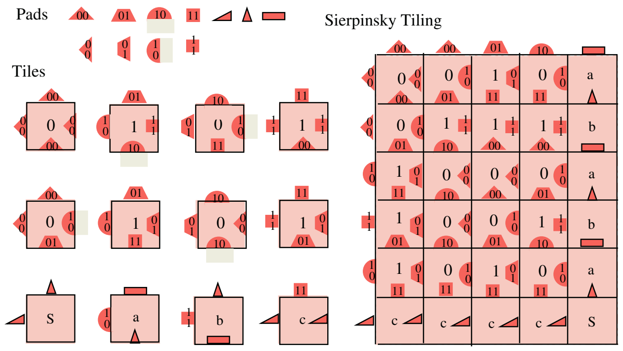


Fig. 8. The construction of a Sierpinsky Triangle using error resilient assemblies version 1. The pads and the tile set are shown on the left and the assembled Sierpinsky Triangle is shown on the right. The pads of strength 2 have black borders while the strength 1 pads are border-less. The seed tile is labeled with S. Tiles a, b, and c are the other frame tiles

error in the assembly in 50% cases. The variations in the value of N are measured as we increase value of the probability of a single mismatch in pads (ϵ) by changing the values of G_{mc} and G_{se}, where G_{mc} and G_{se} are the free energies [12]. We used the fact mentioned by Winfree [12] that $\epsilon \approx 2e^{-G_{se}}$ and for a good assembly we need to have $G_{mc} \approx 2G_{se}$.

Figure 9 shows the variation in N with $\log_e \epsilon$. From the figure it can be seen that the performance of our version 1 (T_1) construction is comparable to Winfree's 2×2 proofreading tile set construction, while our version 2 (T_2) performs comparably to Winfree's 3×3 proofreading tile set construction.

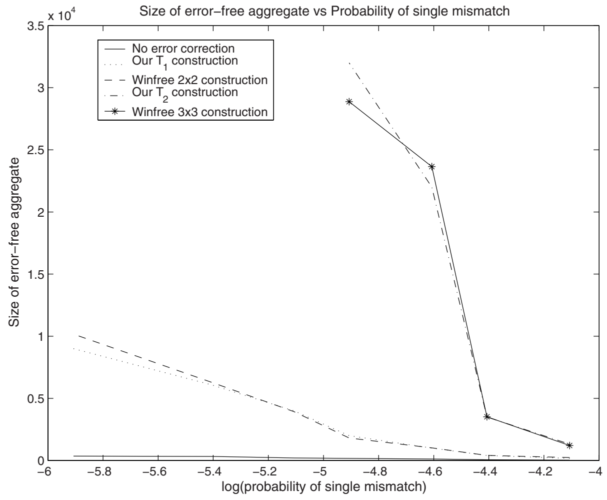


Fig. 9. A graph showing the variation of N *v.s.* increasing value of error (probability of single mismatch)ϵ

7 Discussion

In the proof of this paper, we require OP_1 to be XOR, for concreteness. However, note that our constructions apply to more general boolean arrays in which OP_1 is an *input sensitive operator*, i.e. the output changes with the change of exactly one input.

Note that OP_1 and OP_2 are both the function XOR for the example assemblies for the Sierpinsky Triangle but this is not true for the assembly for a binary counter of N bits, since OP_2 is the logical AND in that example. It is an open question whether our above error-resilient constructions can be further simplified in the case of special computations, such as the Sierpinsky Triangle, where the OP_1 and OP_2 are the same function such as XOR.

Another open question is to extend the construction into a more general construction such that the error probability can be decreased to ϵ^k for any given k, or alternatively, prove an upper bound for k.

Acknowledgements

We would like to thank Erik Winfree for the simulation software Xgrow. We are also grateful to Thomas H. LaBean and Hao Yan for helpful discussions. This work was supported by NSF under ITR Grant EIA-0086015 and ITR Grant 0326157, by NSF under QuBIC Grant EIA-0218376 and QuBIC Grant EIA-0218359, by NSF under EMT Grant CCF-0432038 and EMT Grant CCF-0432047, by DARPA/AFSOR under Contract F30602-01-2-0561, and by RGC under Grant HKBU2107/04E.

References

1. B. A. Bondarenko. *Generalized Pascal Triangles and Pyramids, Their Fractals, Graphs and Applications*. The Fibonacci Association, 1993. Translated from Russian and edited by R. C. Bollinger.
2. H. L. Chen, Q. Cheng, A. Goel, M. D. Huang, and P. M. de Espanes. Invadable self-assembly: Combining robustness with efficiency. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2004.
3. T. H. LaBean, H. Yan, J. Kopatsch, F. Liu, E. Winfree, J. H. Reif, and N. C. Seeman. The construction, analysis, ligation and self-assembly of DNA triple crossover complexes. *J. Am. Chem. Soc.*, 122:1848–1860, 2000.
4. M. G. Lagoudakis and T. H. LaBean. 2-D DNA self-assembly for satisfiability. In *DNA Based Computers V*, volume 54 of *DIMACS*, pages 141–154. American Mathematical Society, 2000.
5. C. Mao, W. Sun, and N. C. Seeman. Designed two-dimensional DNA holliday junction arrays visualized by atomic force microscopy. *J. Am. Chem. Soc.*, 121:5437–5443, 1999.
6. J. H. Reif. Local parallel biomolecular computation. In H. Rubin and D. H. Wood, editors, *DNA-Based Computers 3*, volume 48 of *DIMACS*, pages 217–254. American Mathematical Society, 1999.
7. J. H. Reif, S. Sahu, and P. Yin. Compact error-resilient computational DNA tiling assemblies. Technical Report CS-2004-08, Duke University, Computer Science Department, 2004.
8. N. C. Seeman. DNA in a material world. *Nature*, 421:427–431, 2003.

9. J. von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. *Autonomous Studies*, pages 43–98, 1956.
10. E. Winfree. On the computational power of DNA annealing and ligation. In R. J. Lipton and E. B. Baum, editors, *DNA Based Computers 1*, volume 27 of *DIMACS*, pages 199–221. American Mathematical Society, 1996.
11. E. Winfree. Simulation of computing by self-assembly. Technical Report 1988.22, Caltech, 1998.
12. E. Winfree and R. Bekbolatov. Proofreading tile sets: logical error correction for algorithmic self-assembly. In *DNA Based Computers 9*, volume 2943 of *Lecture Notes in Computer Science*, pages 126–144, 2004.
13. E. Winfree, F. Liu, L. A. Wenzler, and N. C. Seeman. Design and self-assembly of two-dimensional DNA crystals. *Nature*, 394:539–544, 1998.
14. E. Winfree, X. Yang, and N. C. Seeman. Universal computation via self-assembly of DNA: Some theory and experiments. In L. F. Landweber and E. B. Baum, editors, *DNA Based Computers II*, volume 44 of *DIMACS*, pages 191–213. American Mathematical Society, 1999.
15. H. Yan, S. H. Park, G. Finkelstein, J. H. Reif, and T. H. LaBean. DNA-templated self-assembly of protein arrays and highly conductive nanowires. *Science*, 301:1882–1884, 2003.

Toward "Wet" Implementation of Genetic Algorithm for Protein Engineering

Kensaku Sakamoto[1], Masayuki Yamamura[2], and Hiroshi Someya[3]

[1] Department of Biophysics and Biochemistry, Graduate School of Science, The University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo 113-0033
and
RIKEN Genomic Sciences Center, 1-7-22 Suehiro-cho, Tsurumi, Yokohama 230-0045, Japan
sakamoto@biochem.s.u-tokyo.ac.jp

[2] Department of Computational Intelligence and Systems Science, Interdisciplinary Graduate School of Science and Engineering, Tokyo Institute of Technology, Nagatsuta 4259, Midori-ku, Yokohama 226-0026, Japan
my@dis.titech.ac.jp

[3] Department of Prediction and Control, The Institute of Statistical Mathematics, 4-6-7 Minami-Azabu, Minato-ku, Tokyo 106-8569, Japan
some@ism.ac.jp

Abstract. We here propose an application of DNA computing to a practical problem, protein engineering, which is difficult to approach by using modern electronic computers. DNA molecules naturally carry the blueprints of proteins. DNA-based processing of this genetic information could give mutant proteins with desired properties. We conceived the use of genetic algorithm for this purpose, and designed an algorithm amenable to DNA-based implementation. The performance of this algorithm was examined on a model fitness landscape by computer experiments. Then, spontaneous DNA recombination during PCR was utilized to embody the crossover operation in the genetic algorithm, preparing for the "wet" implementation of the whole search process in the future.

1 Introduction

Adleman's study of DNA computer indicated that the information carried by DNA molecules can be processed by a defined mathematical algorithm, which is implemented using molecular biology techniques (Adleman, 1994). His way of solving a combinatorial problem involves the generation of a library of solution candidates and the search for the true solution. The solution was actually found by a rigorous implementation of a computing algorithm, not by trial and error.

DNA molecules in nature carry the blueprints (genes) of proteins. It is known that some mutant genes (the genes with base substitutions or mutations in their base sequence) produce mutant proteins with an enhanced activity or an altered substrate specificity. In molecular biology, such desired mutant genes have been obtained by exploring an artificially generated library of mutants. This search process could be regarded as a computation of an optimization problem by using biological or "wet" techniques[1]. However, this process has been implemented largely by trial and error, rather than according to an appropriate algorithm.

In order to design such a search algorithm for "evolutionary protein engineering", we could take advantage of the empirical knowledge and conventional tools (computer experiments, for example) of computer scientists, if protein evolution is actually an instance of optimization problem, and the desired mutant genes can be assumed to be optima or sub-optima on the fitness landscape. In the present study, we first surveyed the existing search strategies used in computer science, before proposing a possible algorithm for protein engineering. The performance of this algorithm was examined by computer experiments, and a wet method for one-point crossover was then developed for a future implementation of the whole search process in a wet manner.

2 Proposed Algorithm for Protein Engineering

2.1 Guidelines for Designing Effective Search Algorithm

Several types of stochastic algorithms have been proposed for optimization processes, and Genetic Algorithm (GA) provides one of the successful optimization strategies working on electronic computers or *in silico*, not restricted to particular fitness landscapes (Goldberg, 1989). We adopted GA as the main strategy for the evolutionary protein engineering, because GA is equipped with all of the genetic operators of selection, mutation, and crossover (or recombination), which are regarded as the main driving forces of protein evolution in nature[2]. The use of GA and evolutionary strategy in DNA computing has been conceived and demonstrated in an instance of mathematical problem (Chen, 1999).

Kita and Yamamura have proposed guidelines for designing GA (Kita, 1999a), and several GAs that satisfy the design guidelines have shown effective performance (Kita, 1998; Kita, 1999b; Tsutsui, 1999; Someya, 2001). One of their recommendations is that "selection operators of GA should be designed so as to gradually narrow the distribution of a population of genes and maintain the diversity of the population to a maximum." A selection operator not equipped with this mechanism for the diversity maintaining often causes a disadvantageous phenomenon called "Premature Convergence". If the diversity rapidly decreases in the early stage of the search, the varieties of the characteristics of the genes in the population are also lost. Thus, crossover does not work as recombination and does not generate novel solutions. Since mutation generates new characteristic

[1] The fitness function for protein, computable on electronic computers, is not known yet.
[2] In this report, we use the terminology of molecular biology rather than that of the conventional GA. For example, the term of "gene" or "base sequence" is used instead of the "chromosome" in GA.

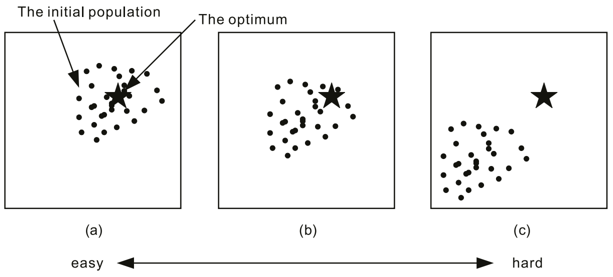


Fig. 1. Relation between the initial population and hardness of finding the optimum

genes, increasing a mutation rate might be a convenient solution to this problem. However, too much mutation often causes another problem called "Evolutionary Stagnation" because it would rather destroy a useful characteristics in a gene than find that one. Note, the above discussions are based on the assumption that the initial population has enough diversity and encompasses the optimum adequately. For example, in the case of (a) in Fig.1, the optimum is located in the center of the initial population, finding the optimum is easy for GA. On the other hand, in the case of (c), GA hardly finds the optimum (Someya, 2002).

2.2 Our Proposed Algorithm

When designing a possible search algorithm for protein engineering, we took into accounts the biological constraints on the search operators, which, on the other hand, can be arbitrarily designed for *in silico* implementations (Table 1). In addition, there is an important difference between the biological and *in silico* searches. In the evolutionary protein engineering, the search starts from the initial gene pool that contains only one or a few gene species, and thus does not satisfy the aforementioned condition that the genes in the initial pool should be so spread in the space as to encompass the desired optimum or optima.

In order to address this difficulty, we employed the mechanism of Simulated Annealing (SA) (Kirkpartrick, 1983), in the scheme of GA, for making a favorable gene diversity at the early stage of search and maintaining it in the succeeding search process. This combination of SA and GA can find its precedents in computer science, such as Thermodynamical Genetic Algorithm (Mori, 1995) and Genetic Simulated Annealing (Koakutsu, 1994). The outline of our proposed algorithm for protein evolution is as the following.

Step 1: Generate the initial population, $P(0)$, including p genes by amplifying the starting gene(s).

Step 2: Introduce mutations into the genes in $P(t)$ to generate $P_m(t)$, where t is the counter of search generations and takes the values of $0, 1, 2, \cdots$

Step 3: Make crossovers between the genes in $P_m(t)$ to generate a population of crossover products, $P_c(t)$.

Step 4: Select a gene from $P_c(t)$ randomly, and accept it at the probability related to both the fitness value of this gene and a given selection pressure

Table 1. Differences between *in silico* and wet implementations

	in silico	in vitro
readability of a gene sequence	easy	hard
automatization	whole	partial
parallel processing	under a few hundred	dozens
designing operators	flexible	constrained
parent-child lineage	observable	not observable

represented by T. This operation is repeated p times to form $P'(t+1)$ of size p'.

Step 5: Select $p - p'$ genes from $P(t)$ randomly, and add them to $P'(t+1)$ to form $P(t+1)$.

Step 6: Go to *Step 2* and set t to be $t+1$, unless a certain condition is satisfied.

The mechanism of SA is employed at *Step 4*. The selection of genes is performed by the Metropolis method, being controlled by parameter T. Suppose that ΔE is the increase in the fitness value for a gene, relative to the average value in the gene population of the previous generation. Then, if ΔE is > 0 or 0 for a gene, this gene is accepted at the probability of 1. If ΔE is < 0, the gene is accepted at the probability of $\exp(\Delta E/T)$. The value for T is gradually decreased throughout the search process, according to a given annealing schedule. This proposed algorithm has the advantages of (i) not requiring the information of base sequence for each gene, (ii) not requiring specific designs for the mutation and crossover operators, (iii) being equipped with the mechanism for tuning the selection pressure, and (iv) being relatively simple.

3 Computer Experiments

3.1 Experimental Details and the Results of Computer Experiments

The performance of an *in silico* search algorithm is usually tested in computer experiments performed on model landscapes, and parameters are adjusted through these trials. Our proposed algorithm was similarly tested for its performance as an optimizer on the landscapes of Kauffman's *NK* model (Kauffman, 1995), a tunable random fitness landscape model. This is not a simulation of a protein evolution based on our algorithm, but a performance test using more general landscapes.

The fitness function is determined as $\frac{1}{N}\sum_{i=1}^{N} f(x_i; x_{j0}, \ldots x_{jK})$, where $f(x_i) \sim U(0,1)$. N is the dimension of the sequence space (length of the gene) and K represents the number of the linkages that each base has with other bases in the same gene. The modality of the fitness landscape is controlled by K. In the case of $K = 0$, the fitness landscape is unimodal and there are no correlation between the bases. In the other cases, $K > 0$, the fitness landscapes

Table 2. Experimental results. r_c indicates crossover rate. The figures represented by bold font specify the best parameter set in each K

			$K=0$	$K=2$	$K=4$	$K=6$	$K=8$
$T(0)$=0.0	No Crossover		100	67.6	16.0	7.8	2.6
	One-point Crossover	$r_c=0.3$	100	72.7	20.4	8.4	2.4
		$r_c=1.0$	100	62.4	13.3	3.8	0.8
	Uniform Crossover	$r_c=0.3$	100	67.5	15.8	6.8	2.2
		$r_c=1.0$	100	45.5	6.2	1.0	0.4
$T(0)$=1.0	No Crossover		100	81.1	22.3	**15.4**	**7.6**
	One-point Crossover	$r_c=0.3$	100	**84.8**	**26.7**	14.2	5.3
		$r_c=1.0$	100	77.6	23.1	7.5	2.4
	Uniform Crossover	$r_c=0.3$	100	71.0	19.5	7.3	4.0
		$r_c=1.0$	100	56.9	9.6	0.2	1.4

have some peaks. As the value for K increases, more number of peaks appear, and the landscape becomes more rugged. The experiments were performed with $K = 0, 2, 4, 6$ or 8. N and the number of different bases for each gene position, A, are set to be 32 and 2, respectively. Five instances of landscape are prepared for each K value. They were randomly generated, and the global optimum for each instance was obtained by exhaustive search in advance.

The initial temperature $T(0)$ is set to be 0.0 or 1.0; with $T(0) = 0.0$, the search will be done in a simple hill-climbing manner, which keeps only the improved genes as the parents of the next generation, while the effects of SA can be observed with $T(0) = 1.0$. The temperature is scheduled based on $T(t+1) = T(t) * 0.98$. The base sequence of the starting gene was generated at random. Population size p is set to be 100. The upper limit on the number of the iteration of *Steps 2* $\sim$ *5* was set to be 2000. One-point crossover and uniform crossover, each with the crossover rate of 0.0, 0.3 or 1.0 (no crossover occurs with the rate of 0.0), were used as the crossover operator, in order to determine which one of these operators gives the highest performance. The mutation rate was fixed to be 0.03125, meaning one mutation (base-type change) at a random site per gene. The performance measure is the success rate of finding the global optimum (r_{opt}). For example, when r_{opt} of a method is 100, this result indicates that the method succeeded in finding the global optimum in all trials. For each instance of landscape, 300 trials were performed, with the results summarized in Table 2. In some searches, the optimum was found out after a small number of search generations. For example, the optimum was found after around 300 generations in the instances for every K value under the conditions of one-point crossover, $r_c = 0.3$, and $T(0) = 1.0$.

Fig.2 shows representative transition curves for the entropy, which corresponds to the gene diversity in a gene population, $\frac{1}{N}\sum_{i=1}^{N}\sum_{a\in A} -P_i(a) \log_2 P_i(a)$, where $P_i(a)$ is the proportion of the genes with base a at the i-th position in the population. In all of the trials starting from $T = 1.0$, the gene diversity increases at the early search generations, up to much higher scores than the trials starting $T = 0.0$.

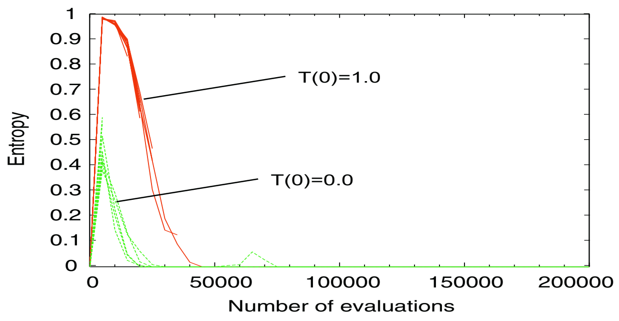


Fig. 2. Representative transition curves for the entropy (gene diversity). These curves were drawn based on the results of 10 trials for an instance with $K = 4$, where the starting temperature $T(0)$ was set to be 0.0 or 1.0, and the one-point crossover was performed at the rate of 0.3. Each search generation involves 100 evaluations. The optimum was found out before 500 search generations in the trials under $T(0) = 1.0$

3.2 Discussion on the Results from Computer Experiments

In the performed computer experiments, the optimization of the fitness value became more difficult, as the value for K increased (each gene position was correlated with more number of other positions in the computation of the fitness value). In the cases of $K = 2$ and $K = 4$, our proposed algorithm, combining the search strategies of GA and SA, showed better performance than the simple SA and the simple hill-climbing. When the value for K was increased up to 6 and 8, the simple SA showed the performance better than, or nearly comparable to, that of our algorithm. As shown in Fig.2 for the instance of $K = 4$, the diversity in the gene population successfully increased for $T(0)=1.0$ at the early stage, which was consistent with the good performance of our algorithm. This successful diversification was concluded to be driven by the mechanism of SA, by comparing with the diversification under $T(0) = 0.0$. These results indicated that our proposed algorithm is a good optimizer on the "correlated" landscapes. In addition, we found that the one-point crossover with a crossover rate of 0.3 is a better operator than the one-point crossover with a rate of 1 or the uniform crossover with any crossover rate. This finding is consistent with the empirical knowledge from *in silico* GA that crossover is to be performed in appropriate manners and with appropriate rates, because it could spread some characteristics in the population so rapidly as to cause premature convergence. On the other hand, the mutation rate was fixed to be one random mutation per gene, and therefore higher or lower mutation rates would possibly give better results.

We note that as many as $300 \sim 2,000$ search generations cannot be implemented in wet experiments. In contrast, in the case of wet implementation, the fitness values of as many as 10,000 to 1,000,000,000 genes could be evaluated at each search generation, by using biological genetic methods. It remains to study whether this heavy parallelism can compensate for a much smaller number of search generations feasible in wet experiments as compared with that feasible *in* *silico*. On the other hand, it seems easy to introduce the mechanism of SA into the evolutionary protein engineering, by keeping genes for the next generation at the probabilities related with their fitness values. The biological embodiment of the one-point crossover was developed in the next section.

4 Wet Embodiment of One-Point Crossover

The biological parallel to crossover is gene recombination, and the feasibility of its implementation in a test tube has been demonstrated (Stemmer, 1994). However, all of the reported methods have been developed for implementing multi-point crossover (Stemmer, 1994; Zhao, 1998). We describe here a method for one-point crossover.

If two genes have some homology between their base sequences, crossover spontaneously occurs during PCR and the succeeding cloning step (Jansen & Ledley, 1990). This crossover during PCR was utilized by Zhao et al., and the frequency of crossover was increased by reducing the time of the polymerization step. On the other hand, if the crossover frequency is made low enough to achieve one-point crossovers, the PCR products undergoing no crossovers will form a large population among all products. Therefore, we added a step of keeping

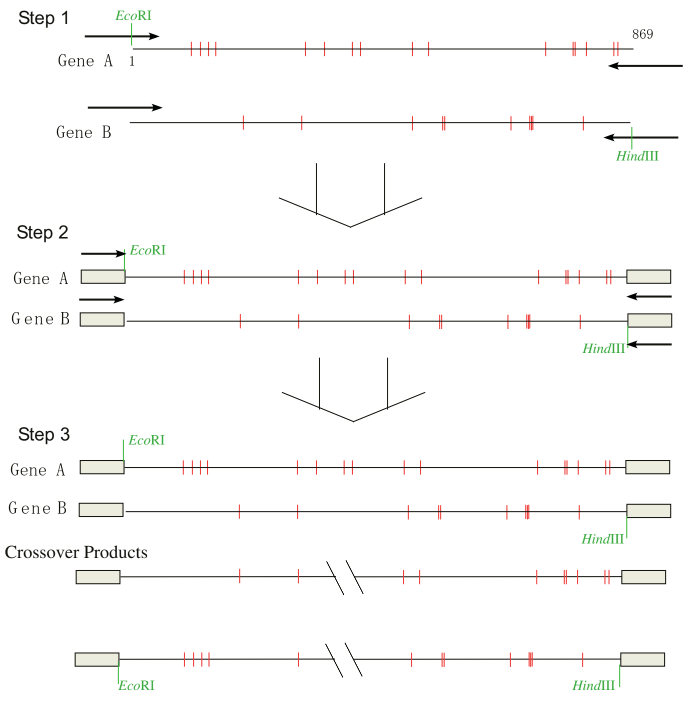


Fig. 3. PCR steps for achieving one-point crossover

Table 3. The number of the crossover products having crossover sites in each section is shown for Programs 1 and 2. The data of only the one-point crossover products are included. Each section is between two neighboring mutation sites, except that positions 1 and 869 are the fist and last positions, respectively, of genes A and B. The positions for the mutations in gene A are underlined, while the others are in gene B except for position 788, at which both genes have different types of base substitution

Section	length	Program 1	Program 2
1-110	109	1	1
-118	8	0	1
-130	12	1	1
-145	15	0	0
-199	54	0	1
-301	102	4	4
-302	1	0	0
-336	34	0	0
-384	48	1	0
-398	14	0	0
-484	86	3	3
-489	5	0	0
-515	26	2	0
-545	30	1	2
-547	2	0	0
-658	111	3	3
-696	38	1	2
-698	2	0	0
-702	4	1	0
-714	12	0	0
-763	49	0	1
-765	2	0	0
-788	23	0	1
-835	47	2	1
-843	8	0	0
-869	26	0	2
total		19	23

only the crossover products, while the crossover at the cloning step was avoided, because this event, occurring in the *E. coli* cell, is difficult to control.

Two genes, A and B, used in the test experiments are the same part of the *E. coli argS* gene (869 bases) with 16 and 10 mutations, respectively (Table 3). Crossover products were generated by the following laboratory steps (Fig. 3).

Step 1: The *Eco*RI and *Hin*dIII sites were generated near the 5' end of gene A and the 3' end of gene B, respectively, by amplifying these genes with appropriate PCR primers. This step was carried out separately for the two genes.

Step 2: The PCR products from the two genes were mixed and then amplified using the common PCR primers. At this step, we tested two PCR programs, Programs 1 and 2, with different times for the polymerization step. The time is shorter in Program 1 than in 2, as described under "Experimental Methods".

Step 3: The products of *Step 2* were diluted and then amplified to less than a concentration of 50-ng products per 1μL of the reaction mixture. This step removes heteoduplex between the two genes, because heteroduplex is formed under high product concentrations achieved at late PCR cycles.

The final PCR products were inserted between the *Eco*RI-*Hin*dIII sites of vector pUC19, and the sequences of 21 and 24 clones for Programs 1 and 2, respectively, were analyzed. All of these clones were crossover products. Although the crossover site could not be pinpointed, we decided which one of the gene sections each between two mutation sites included the crossover site (Table 3). Except for two clones for Program 1 and one for 2, the crossover products were formed by one-point crossover, with the crossover sites distributed along the full-length of the gene.

Our protocol does not completely exclude the contamination of multi-point crossover products. Programs 1 and 2 formed two three-point-crossover products and one such product, respectively; these contaminations were 9.5% for Program 1 and 4.2% for Program 2. The frequency of the occurrence of mutation in the analyzed clones was 0.18% and 0.14% for Programs 1 and 2, respectively.

We also carried out a similar experiment without performing the extraction of crossover products (*Step 1* was omitted), and the sequences of four clones out of the obtained products were analyzed. All of the four were either gene A or B, no crossover products between them. Thus, the population of the spontaneous crossover products contained in the PCR products was small.

In this experiment, all of the crossover products had the 5' and 3' parts of gene A and gene B, respectively. When this method is applied to a population of genes, the *Eco*RI site will be added to some copies of a gene and the *Hin*dIII site to the other copies of this gene. Therefore, both of its 5' and 3' parts will be found in crossover products. One-point crossover guarantees that any product has just two parent genes, while the multi-point crossover performed in a wet manner necessarily generates the crossover products each made of parts from several parent genes.

Experimental Methods. At *Step 1*, gene A or B of more than 5 ng was amplified with ExTaq DNA polymerase (Takara, Japan) of 2 units and appropriate primers (25 pmol each) in a 50-μL reaction. The thermal program comprises pre-heating at 94˚C for 30 sec, 25 cycles [94˚C for 30 sec, 60˚C for 30 sec, and 72˚C for 60 sec], and the graduating run at 72˚C for 3 min. PCR at *Step 3* was similarly performed, except that the products of *Step 2* (50 ng) were amplified by 8 thermal cycles. At *Step 2*, the products of *Step 1* were mixed and then subjected to PCR in a 50-μL reaction using rTaq DNA polymerase (Toyobo, Japan) of 2 units and appropriate primers (25 pmol each). The thermal program consists of a pre-heating at 94˚C for 30 sec, 25 cycle [94˚C for 60 sec, 45˚C for 60 sec, and 72˚C for 30 sec (Program 1) or 60 sec (Program 2)], and the graduating 3-min run at 72˚C . A GeneAmp PCR system 9700 (Applied Biosystems) was used, with a ramp speed set to 9600.

5 Conclusion

The development of DNA computer into a useful tool requires the scaling-up of computations or its application to any purpose that is difficult to approach even with modern electronic computers. DNA computers operating on undefined DNA sequences have been conceived as a possible application of the latter category (Landweber, 1999). DNA-based GA for protein engineering also operates on the undefined sequences produced by random mutation and recombination, and the protein design by electronic computers has not yet enjoyed wide success. Thus, our study explores the new way to use DNA computer in its application to biotechnology.

Acknowledgement

We thank H. Kita and Y. Husimi for valuable discussions. This work was supported by the Grant-in-Aid for Scientific Research on the Priority Area "Molecular Implementation and Methodology for Natural Computing (in Molecular Programming) (No. 766-14085203)" by the MEXT of Japan.

References

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

Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison Wesley (1989)

Chen, J., Antipov, E., Lemieux, B., Cedeno, W., Wood, D.H.: *In vitro* selection for max 1s DNA genetic algorithm. In: Preliminary Proceedings of 5th International Meeting on DNA Based Computers. (1999) 23–37

Kita, H., Yamamura, M.: A functional specialization hypothesis for designing genetic algorithms. In: Proceedings of 1999 IEEE International Conference on Systems, Man and Cybernetics. (1999) 579–584

Kita, H., Ono, I., Kobayashi, S.: Theoretical analysis of the unimodal normal distribution crossover for real-coded genetic algorithms. In: Proceedings of 1998 International Conference on Evolutionary Computation. (1998) 529–534

Kita, H., Ono, I., Kobayashi, S.: Multi parental extension of the unimodal normal distribution crossover for real-coded genetic algorithms. In: Proceedings of the 1999 Congress on Evolutionary Computation. (1999) 1581–1587

Tsutsui, S., Yamamura, M., Higuchi, T.: Multi-parent recombination with simplex crossover in real coded genetic algorithms. In: Proceedings of the Genetic and Evolutionary Computation Conference 1999. (1999) 657–664

Someya, H., Yamamura, M.: Genetic algorithm with search area adaptation for the function optimization and its experimental analysis. In: Proceedings of the 2001 Congress on Evolutionary Computation. (2001) 933–940

Someya, H., Yamamura, M.: Robust evolutionary algorithms with toroidal search space conversion for function optimization. In: Proceedings of the Genetic and Evolutionary Computation Conference 2002. (2002) 553–560

Kirkpatrick, S., Gelatt, C., Vecchi, M.: Optimization by simulated annealing. Science **220** (1983) 671–680

Mori, N., Yoshida, J., Tamaki, H., Kita, H., Nishikawa, Y.: A thermodynamical selection rule for the genetic algorithm. In: Proceedings of 1995 International Conference on Evolutionary Computation. (1995) 188–192

Koakutsu, S., Hirata, H.: Genetic simulated annealing for floorplan design. Control and Information Sciences **197** (1994) 268–277

Kauffman, S.A., Macready, W.G.: Search strategies for applied molecular evolution. Journal of Theoretical Biology **173** (1995) 427–440

Stemmer, W.P.C.: Rapid evolution of a protein in vitro by DNA shuffling. Nature **379** (1994) 389–391

Jansen, R., Ledley, F.D.: Disruption of phase during pcr amplification and cloning of heterozygous target sequences. Nucleic Acids Research **18** (1990) 5153–5156

Zhao, H., Giver, L., Affholter, J.A., Arnold, F.H.: Molecular evolution by staggered extension process (StEP) in vitro recombination. Nature biotechnology **16** (1998) 258–261

Landweber, L.F., Lipton, R.J., Rabin, M.O.: DNA^{2}DNA computations: a potential “Killer App?”. DIMACS Series in Discrete Mathematics and Theoretical Computer Science **48** (1999) 161–172

Programmable Control of Nucleation for Algorithmic Self-assembly

(Extended Abstract[*])

Rebecca Schulman and Erik Winfree

California Institute of Technology,
Pasadena, CA 91125, USA
{rebecka, winfree}@caltech.edu

Abstract. Algorithmic self-assembly has been proposed as a mechanism for autonomous DNA computation and for bottom-up fabrication of complex nanodevices. Whereas much previous work has investigated self-assembly programs using an abstract model of irreversible, errorless assembly, experimental studies as well as more sophisticated reversible kinetic models indicate that algorithmic self-assembly is subject to several kinds of errors. Previously, it was shown that proofreading tile sets can reduce the occurrence of mismatch and facet errors. Here, we introduce the zig-zag tile set, which can reduce the occurrence of spurious nucleation errors. The zig-zag tile set takes advantage of the fact that assemblies must reach a critical size before their growth becomes favorable. By using a zig-zag tile set of greater width, we can increase the critical size of spurious assemblies without increasing the critical size of correctly seeded assemblies, exponentially reducing the spurious nucleation rate. In combination with proofreading results, this result indicates that algorithmic self-assembly can be performed with low error rates without a significant reduction in assembly speed. Furthermore, our zig-zag boundaries suggest methods for exquisite detection of DNA strands and for the replication of inheritable information without the use of enzymes.

1 Introduction

Since Adleman first used DNA to perform a hard computation [1], researchers have explored the ability of biological molecules to carry out algorithms. Algorithmic self-assembly of DNA tiles [19] is Turing universal in theory, and tile sets for the construction of a variety of desired products have been suggested [12, 15, 3, 8]. An example of a structure that can be constructed using algorithmic self-assembly, a Sierpinski triangle, is shown in Figure 1. A simple generalization of this construction can be used to implement an arbitrary cellular automaton.

A tile program consists of labels for the sides of each of a set of square tiles, the strength with which each possible pair of labels binds, a designated seed tile,

[*] A preprint of the full paper can be found at `http://arxiv.org`.

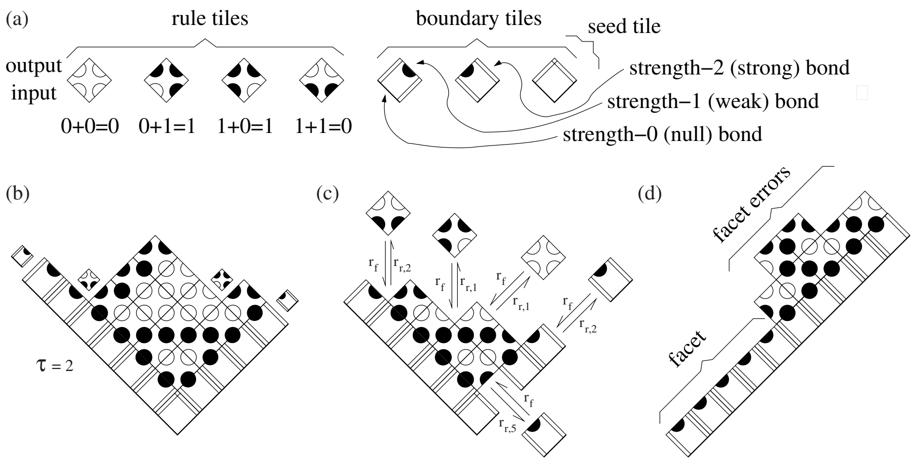


Fig. 1. Tile assembly models (a) The Sierpinski tile set. Tiles cannot be rotated. Whereas the rule tiles have sides that form strength-1 (weak, single-line) bonds, some sides on the boundary tiles form strength-2 (strong, double-line) bonds or strength-0 (null, thick-line) bonds. **(b)** Seeded growth of the Sierpinski tiles according to the aTAM at $\tau = 2$. The small tiles indicate the (only) four sites where growth can occur. At each location exactly one tile matches both exposed sides, so assembly results in a unique pattern. **(c)** For the growth of an isolated crystal under unchanging tile concentrations, the forward rate (association) is $r_f = k_f[tile] = k_f e^{-G_{mc}}$, while the reverse rate (dissociation) is $r_{r,b} = k_f e^{-bG_{se}}$ for a tile that makes bonds with total strength b. Parameters G_{mc} and G_{se} govern monomer tile concentration and sticky-end bond strength, respectively. A representative selection of possible events is shown here. The kTAM approximates the aTAM with threshold τ when $G_{mc} = \tau G_{se} - \epsilon$, in which case the same set of reactions are favorable or unfavorable in the two models. **(d)** An undesired assembly that can form due to unseeded growth of boundary tiles followed by facet errors

and a strength threshold τ. Polyomino tiles with labels on each unit-length of the perimeter can be used in addition to square tiles. The abstract tile-assembly model (aTAM) [20] describes the behavior of a tile program executed in the absence of assembly errors. Under the aTAM, assembly starts with designated tiles (usually just the seed tile) and proceeds by the addition of tiles at locations on the assembly's perimeter where the total strength of the connections between the tile and the assembly is greater than or equal to the threshold. Addition of tiles is irreversible but non-deterministic. Within the aTAM, it is possible to prove program correctness – that is, that growth from the seed tile always results in the unique desired structure. In this paper, unless stated otherwise mismatched labels will always bind with strength 0, bond strengths will be non-negative integers, and $\tau = 2$, as is the case for most prior work on algorithmic self-assembly.

In contrast to assembly in the aTAM, the assembly of DNA tiles is neither errorless nor irreversible. In practice a tile sometimes binds and sticks to a growing assembly even when the strength of the tile's attachments is smaller than the threshold, an event called an *unfavorable attachment.* Unfavorable attachments can lead to three kinds of errors. First, an unfavorable attachment that only partially matches the adjacent tiles can occasionally become locked into place by succeeding attachments, forming a mismatch error. Second, a tile that attaches unfavorably to a facet and in turn allows the attachment of incorrect tiles nearby causes a facet error (Figure 1(d)). Lastly, a spurious nucleation error occurs when an assembly grows from a tile other than the designated seed tile.

Mismatch, facet and spurious nucleation errors have all been observed in algorithmic self-assembly experiments. In an experimental demonstration of the algorithmic self-assembly of a Sierpinski triangle [16], between 1% and 10% of tiles mismatched their neighbors, an effect that was attributed to both mismatch and facet errors. Furthermore, only a small fraction of the observed crystals were properly nucleated from seed molecules.

Why avoiding spurious nucleation can be difficult was clarified by experiments with just the boundary tiles shown in Figure 1 [17]. While the aTAM predicts that V-shaped assemblies should form, most observed assemblies were linear polymers without a seed tile. When all the tiles in the Sierpinski tile set were combined, most assemblies seen were spuriously nucleated rather than grown from a V-shaped boundary. The spuriously nucleated assemblies could have been produced either by linear boundaries growing wider due to multiple facet errors or by rule tiles assembling by themselves into crystals.

To theoretically study the rates at which these three kinds of errors occur, we need a model that includes energetically unfavorable events. The kinetic Tile Assembly Model (kTAM) [20] describes the dynamics of assembly according to an inclusive set of reversible chemical reactions: A tile can attach to an assembly anywhere that it makes even a weak bond, and any tile can dissociate from the assembly at a rate dependent on the total strength with which it adheres to the assembly (see Figure 1(c)). Several variants of the kTAM, reflecting different assumptions about how growth proceeds, have been developed [18]. In the kTAM as described in Figure 1(c), mismatch errors occur at least at a rate proportional to the square root of assembly speed [20]. Therefore, the mismatch error rate can be reduced by decreasing the temperature of the assembly reaction and/or decreasing the monomer concentration – but a 10-fold decrease in error rates requires a 100-fold decrease in assembly speed. A better solution to controlling mismatch and facet errors is to use "proofreading" tile sets that implement the same logic of an original tile set but assemble more robustly, reducing the error rates exponentially without significant slow-down [21, 6, 14].

In this paper we propose a method to control spurious nucleation errors without significant slow-down, exponentially reducing the rate at which assemblies without a seed tile grow large (unseeded growth), while maintaining the rate of growth that starts from a seed tile and proceeds roughly according to aTAM (seeded growth). To do so, a tile set must satisfy two conflicting constraints: When assembly begins from a seed tile, it must proceed quickly, whereas when assembly starts from a non-seed tile, it must go nowhere.

These two constraints are simultaneously satisfied by a phenomenon well-known to children who make rock candy: the nucleation of crystals in a supersaturated solution. By cooling a solution slowly, it is possible to create a solution that has more solute dissolved than would be possible in standard conditions, called a supersaturated solution. Because of the interplay between surface and volume energy terms in a supersaturated solution, crystals smaller than a critical size will tend to shrink, whereas large crystals will grow. The seeded growth of crystals results from the mixing of a supersaturated solution with a small number of large crystals, called seed nuclei. When a seed nucleus is added to the solution, its growth is immediately favorable. Monomers attach to the seed, and a large crystal results.

To apply these principles for the control of nucleation in algorithmic crystals, it is enough to create a well-behaved boundary that plays the same role as the V-shaped boundary in Figure 1, but grows exclusively from a seed. Since rule tiles are not likely to spuriously nucleate on their own under optimal assembly conditions, once the well-behaved boundary has set up the correct initial information, algorithmic crystal growth will proceed correctly and without spurious side products. We use large seed tiles that serve the the same purpose as the large seed nuclei in the rock candy example. Tiles attach to the seed tile to produce a long boundary of predefined width. Because only full-width boundaries can grow by favorable attachments, without the seed tile there is a critical size barrier that prevents spurious nucleation – unlike the boundary tiles of Figure 1 for which the critical size is a single tile. The tile set that implements these ideas, called the zig-zag tiles, is described below.

2 The Zig-Zag Tile Set

The **zig-zag tile set** (see Figure 2(a)) of width k contains tiles that assemble to form a periodic ribbon of width k (see Figure 3(a)). Zig-zag tile sets can be constructed with any width $k \geq 2$. A zig-zag tile set includes a **top tile** and a **bottom tile**, each consisting of 2 horizontally connected square tiles. It also includes an L-shaped **seed tile** consisting of k vertically connected square tiles and a square tile horizontally connected to the bottom of the vertical connected tiles. In a zig-zag assembly, the top and bottom tiles stagger so that each column of tiles is connected to the columns on its right and left by either a top tile or a bottom tile. Each of the $k-2$ rows between the top and bottom tiles contains two unique **middle tiles** that alternate horizontally. Unique tiles in each row make assemblies of width less than k impossible to form without zero-strength attachments. Two tile types in each row enforce the staggering of the top and bottom tiles, which is essential for seeded growth to proceed quickly in a path that zig-zags up and down the width of the assembly. The seeded assembly path is shown in Figure 3(b).

The tile set is designed to operate in a physical regime where the attachment of a tile to another tile or assembly by two matching sides is energetically favorable, whereas an attachment by only one bond is energetically unfavorable. In

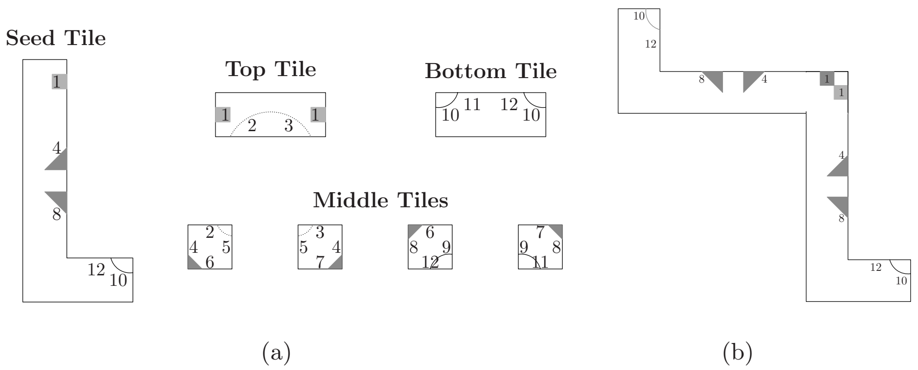


Fig. 2. The zig-zag tile set. (a) Each square, rectangle, and L shape represents a single tile. Excluding the seed tile, tiles are given unique bonds that determine where the fit in the assemble: each label has exactly one match on another tile. All correctly-matched bonds have strength 1. The geometric patterns shown on each tile identify them in subsequent figures. **(b)** The seed shown here, with appropriate tiles for vertical zig-zag growth, could be used instead of the L-shaped seed in (a) to form V-shaped assemblies

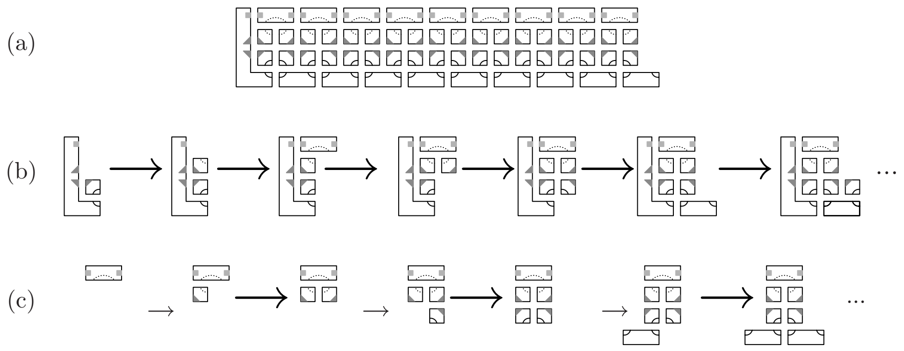


Fig. 3. Zig-zag assembly. (a) The structure formed by the zig-zag tile set according to aTAM with a threshold $\tau = 2$. **(b)** Seeded growth of a zig-zag tile set in the aTAM. The same growth pattern occurs reversibly in the kTAM with a threshold near $\tau = 2$. **(c)** A possible series of steps by which the tiles could spuriously nucleate in the kTAM. Under the conditions of interest, some steps are energetically favorable, but at least $k - 1$ must be unfavorable for a zig-zag tile set of width k. At this point, further growth is favorable

this physical regime, algorithmic self-assembly is possible. In the aTAM, these conditions translate to growth with a threshold of 2. Growth from a seed tile occurs in a zig-zag shaped pattern; if assembly starts from a non-seed tile, no growth occurs. In the kTAM, seeded growth occurs in the same pattern as in the aTAM, but there are also series of reactions that can cause spurious nucleation errors.

Spurious nucleation is a transition from assembly *melting*, where assemblies are more likely to fall apart than they are to get larger, to assembly *growth*, where each assembly step is energetically favorable. An assembly where melting and growth are both energetically favorable is a **critical nucleus**. Nucleation theory [9] predicts that the rate of nucleation is limited by the concentration of the critical nucleus, $[A_c]$. Since $[A_c] = e^{-\Delta G/kT}$, where ΔG is the free energy of a critical nucleus with respect to unbound tiles, linearly increasing the energy barrier, ΔG, exponentially decreases the rate of nucleation[1].

Since there is no energy barrier to seeded growth in the zig-zag tile set, growth from the seed tile is favorable. In contrast, there is an energy barrier for unseeded growth. The size of this barrier depends on the total concentration of critical nuclei. For a zig-zag tile set of width k, the critical nuclei are k tiles wide. Under the right conditions, the energy barrier depends linearly on the width of the critical nuclei, and thus the concentration of critical nuclei decreases exponentially with k. This argument is not rigorous, however, because unfortunately there are also many more kinds of critical nuclei for larger values of k. The rate of spurious nucleation is proportional to the sum of the concentrations of all these critical nuclei.

To bound rather than explicitly calculate the rate of spurious nucleation, it is not necessary to calculate the rate of growth of each critical nucleus. Instead, we consider a set of subcritical assemblies, and we bound the total flux of assemblies leaving this set; it is assumed that (in the worst case) every assembly that leaves the set eventually becomes a long spuriously nucleated ribbon. This flux rate is a valid upper bound as long as single tiles are members of the set and spuriously nucleated assemblies are not.

For the zig-zag tile set of width k, we use the set of assemblies of width less than k. Because of the way the zig-zag tile set is designed, no assembly of width smaller than k can grow significantly longer without an unfavorable attachment. However, any assembly of width k can grow in a zig-zag fashion by exclusively favorable steps. Thus, we bound the rate of spurious nucleation by the rate at which assemblies of width $k-1$ grow to a width of k.

To formally calculate such a bound, we make use of the kTAM as formulated for mass action dynamics [18], assuming constant tile concentrations. In mass-action dynamics the rate at which a reaction proceeds is proportional to a rate constant times the product of the concentrations of the reactants [10]. Given a tile set, we consider all possible accretion reactions: reactions either between two tiles or between a tile and an assembly in which tile concentrations remain constant. Since changes in the concentrations of unbound tiles are ignored[2], a reaction's rate is dependent on at most one changing concentration, so dynamics are linear and therefore easier to analyze. The concentration of tiles and the strength of formation are specified by the parameters G_{mc} and G_{se}. The concentration of

[1] In the kTAM, $\Delta G = (bG_{se} - nG_{mc})kT$ for an assembly involving n tiles and total bond strength b. k is Boltzmann's constant and T absolute temperature.

[2] In reality, tile concentrations will decrease as they are used, further decreasing the rate of spurious nucleation.

each tile (except the seed tile) is $[tile] = e^{-G_{mc}}$ and the bond strength between two matching tiles is G_{se}. The rate constant for each possible forward reaction is k_f, and the reverse rate constant for a reaction involving b bonds is $k_f e^{-bG_{se}}$. For a zig-zag tile set of width k, $J(k)$ is defined as the total rate of all addition reactions that exit the set of subcritical assemblies, i.e., reactions in which the reactant has width $k-1$ and the product has width k. We have proved the following theorem:

Theorem 1. *For a zig-zag tile set of width $k > 2$, if $G_{se} > 2(k \ln 2 + 1)$, $G_{mc} = 2G_{se} - \epsilon$, and $0 \leq \epsilon < \frac{1}{k}$, then, at all time points, $J(k) < 4k_f e^{\epsilon - kG_{se}}$.*

The proof appears in the full paper.

3 Discussion

3.1 Nucleation of Algorithmic Self-assembly

Our original motivation for this work was to show that self-assembly programs that work in the aTAM, in which it is straightforward to design tile sets that algorithmically assemble any computationally defined structure, can also be made to work in the more realistic kTAM. Tiles sets that assemble correctly *via* unseeded growth in the aTAM with a threshold of $\tau = 1$ will assemble correctly in the kTAM under the right conditions. However, tile sets that are designed to assemble *via* seeded growth in the aTAM with a threshold $\tau = 2$ may fail in the kTAM because mismatch, facet and spurious nucleation errors occur. These problems are ameliorated in the limit of slow assembly speed [20]. Other work has described methods to control mismatch errors and facet errors without significant slowdown [21, 6, 14]. Here, we have developed a construction that corrects the last discrepancy, spurious nucleation errors, again without significant slowdown.

However, it remains to be formally proven that these constructions can be combined to control all types of errors simultaneously for any tile set of interest. No major difficulties are expected, in large part because mismatch and facet errors can both be controlled by a single mechanism [6] and the control of spurious nucleation errors works independently of this mechanism. Both methods work by transforming an original tile set which works in the aTAM at $\tau = 2$ into a new (typically larger) tile set that is more robust to particular kinds of errors in the kTAM. The transforms are simple : each tile in the original tile set is replaced by a $k \times k'$ block of tiles with a specified pattern of labels that implement the original tile's logic. The proofreading methods [21, 6, 14] transform rule tiles, while the zig-zag tile set can be considered a transform of the seed and boundary tiles. The cost of both these transformations is a moderate increase in spatial scale and the number of tile types.

As an alternative to these methods, one might wonder whether it is possible to also design tile sets capable of any desired computation that rely only on unseeded growth, which appears to be easier to implement experimentally. However, seeded growth, and therefore control of nucleation, appears to be necessary for practical, algorithmic construction by self-assembly: The seed sets up the

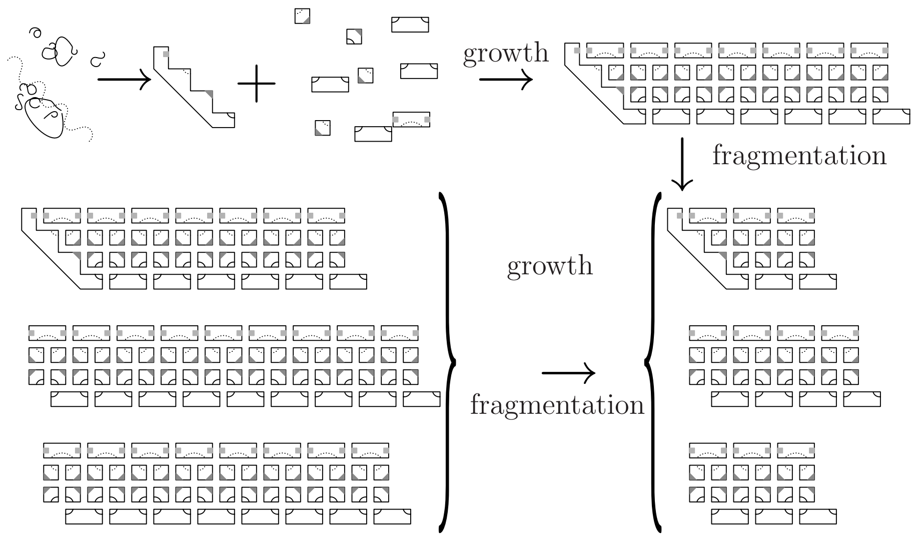


Fig. 4. Exponential amplification of assemblies. Probe strands assemble onto a target sequence to create a seed assembly, which nucleates zig-zag growth. Periodic fluid shear causes fragmentation of zig-zag assemblies, leading to exponential amplification. The diagonal structure of the seed assembly shown here is the natural shape for assembling DAE-E tiles on a scaffold strand [16]

correct initial inputs and directs computation to proceed from beginning to end. (As a consequence, existing mismatch and facet error correction techniques have only been shown to reduce errors in properly seeded assemblies.) Unseeded growth is much more difficult to program and to analyze than is seeded self-assembly, because the "computation" can begin in the middle and proceed in either direction. Although it is possible to assemble computationally defined sets of structures using unseeded growth [1, 22, 4], we would not expect them to assemble efficiently a set of structures as rich as that generated by seeded self-assembly.

3.2 Exquisite Detection of DNA Sequences

Control over nucleation in algorithmic self-assembly can be seen as a special case of exquisite detection (the detection of a single molecule) [2]. For a tile set of sufficiently large width, essentially nothing happens when no seed tiles are present, whereas if even a *single* seed tile is added, growth by self-assembly will result in a macroscopic assembly. Theorem 1 shows that the *false-positive* rate for detection can be made arbitrarily small by design; the *false-negative* rate in the kTAM is 0. Although this idealized model does not consider many factors that could lead to poorer detection in a real system, we don't know of any insurmountable problems with implementing exquisite detection.

There are, however, two immediate drawbacks. First, detecting seed-tile assemblies is not as useful as detecting arbitrary DNA sequences. Second, the linear growth of a single zig-zag assembly would require a long time lapse before a macroscopic change is perceptible. As sketched in Figure 4, we can surmount both obstacles. First, as in [13, 23], a set of strands can be designed to assemble double-crossover molecules on a (sufficiently long) target strand with nearly arbitrary sequence, thus creating the seed assembly if and only if the target strand exists. Second, since fluid shear forces can fragment large DNA assemblies, intermittent pipetting or vortexing will break large zig-zag assemblies, thus at least doubling the number of growing ends with each fragmentation episode. This fragmentation process can be expected to lead to exponential growth in the number of zig-zag assemblies without increasing the false-positive rate.[3]

3.3 Exponential Replication of Inheritable Information

The zig-zag constructions detailed in this paper propagate a single bit of information: the presence or absence of the seed tile. Using a tile set that simply copies information, we could use the exponential amplification reaction to detect and identify one of several different target strands, by creating a tile set where the seed assemblies for each target strand contain a different pattern of 1s and 0s.

Furthermore, considering the amplification process as replication, the information encoded in the strip's width can be seen as a form of inheritable information. A zig-zag assembly replicates (in the appropriate culture medium consisting of tiles) by growth of new layers followed by random fission [11]. Errors during growth, bit flips as well as errors that increase or decrease the width of the assembly, are inherited. If one sequence of tiles has a greater reproductive fitness than other sequences – for example, by having a different growth or fission rate – then natural evolution can be expected to occur. Cairns-Smith considered related ideas about crystal growth as a possible scenario for the origin of life on Earth [5]. However, additional mechanisms would have to be present for this inheritable information to be useful in directing the reproduction of tile sequences. Such an enzyme-free system would be considerably less complex than that controlling the replication of chemical information in modern biological organisms or in processes such as polymerase chain reaction (PCR) that provide the basis for most *in vitro* evolution studies.

Acknowledgment. The authors are grateful to Ho-Lin Chen, Ashish Goel, Rizal Hariadi, Paul Rothemund, Bernie Yurke, and Dave Zhang for helpful advice and discussions. This work was supported by NSF CAREER Grant No. 0093486 to EW and an NSF Graduate Fellowship to RS.

References

1. L. M. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266:1021–1024, Nov. 11, 1994.
2. L. M. Adleman. Personal communication, 2004.

[3] When a spuriously nucleated assembly does eventually form, of course, it will also be exponentially amplified.

3. L. M. Adleman, Q. Cheng, A. Goel, and M.-D. Huang. Running time and program size for self-assembled squares. *Symposium on the Theory of Computing (STOC)*, 2001.
4. L. M. Adleman, J. Kari, L. Kari, and D. Reishus. On the decidability of self-assembly of infinite ribbons. *Symposium on Foundations of Computer Science (FOCS)*, 43:530, 2002.
5. A. G. Cairns-Smith. *The life puzzle: on crystals and organisms and on the possibility of a crystal as an ancestor.* Oliver and Boyd, New York, 1971.
6. H.-L. Chen and A. Goel. Error free self-assembly using error prone tiles. Accepted to *DNA Computing 10.*
7. J. Chen and J. Reif, editors. *DNA Computing 9*, volume LNCS 2943, Berlin Heidelberg, 2004. Springer-Verlag.
8. M. Cook, P. W. K. Rothemund, and E. Winfree. Self-assembled circuit patterns. In Chen and Reif [7], pages 91–107.
9. R. Davey and J. Garside. *From Molecules to Crystallizers.* Oxford University Press, Oxford, UK, 2000.
10. K. A. Dill and S. Bromberg. *Molecular Driving Forces: Statistical Thermodynamics in Chemistry and Biology.* Garland Science, 2002.
11. G. Egan. Wang's carpets. In G. Bear, editor, *New Legends.* Legend, London, 1995.
12. M. G. Lagoudakis and T. H. LaBean. 2-D DNA self-assembly for satisfiability. In E. Winfree and D. K. Gifford, editors, *DNA Based Computers V*, volume 54 of *DIMACS*, pages 141–154, Providence, RI, 2000. American Mathematical Society.
13. C. Mao, T. H. LaBean, J. H. Reif, and N. C. Seeman. Logical computation using algorithmic self-assembly of DNA triple-crossover molecules. *Nature*, 407(6803):493–496, 2000.
14. J. H. Reif, S. Sahu, and P. Yin. Compact error-resilient computational DNA tiling assemblies. Accepted to *DNA Computing 10.*
15. P. W. K. Rothemund. Using lateral capillary forces to compute by self-assembly. *Proceedings of the National Academy of Sciences*, 97:984–989, 2000.
16. P. W. K. Rothemund, N. Papadakis, and E. Winfree. Algorithmic self-assembly of DNA Sierpinski triangles. 2004. Submitted.
17. R. Schulman, S. Lee, N. Papadakis, and E. Winfree. One dimensional boundaries for DNA tile self-assembly. In Chen and Reif [7], pages 108–125.
18. R. Schulman and E. Winfree. Kinetic models of DNA tile self-assembly. In preparation.
19. E. Winfree. On the computational power of DNA annealing and ligation. In R. J. Lipton and E. B. Baum, editors, *DNA Based Computers*, volume 27 of *DIMACS*, pages 199–221, Providence, RI, 1996. American Mathematical Society.
20. E. Winfree. Simulations of computing by self-assembly. Technical Report CS-TR:1998.22, Caltech, 1998.
21. E. Winfree and R. Bekbolatov. Proofreading tile sets: Error-correction for algorithmic self-assembly. In Chen and Reif [7], pages 126–144.
22. E. Winfree, X. Yang, and N. C. Seeman. Universal computation via self-assembly of DNA: Some theory and experiments. In L. F. Landweber and E. B. Baum, editors, *DNA Based Computers II*, volume 44 of *DIMACS*, pages 191–213, Providence, RI, 1998. American Mathematical Society.
23. H. Yan, T. H. LaBean, L. Feng, and J. H. Reif. Directed nucleation assembly of DNA tile complexes for barcode-patterned lattices. *Proc. Nat. Acad. Sci. USA*, 100(14):8103–8108, 2003.

DNA Hybridization Catalysts and Catalyst Circuits

Georg Seelig[1], Bernard Yurke[1,2], and Erik Winfree[1]

[1] California Institute of Technology, Pasadena, CA 91125, USA
[2] Bell Laboratories, Murray Hill NJ 07974, USA
{seelig, yurke, winfree}@dna.caltech.edu

Abstract. Practically all of life's molecular processes, from chemical synthesis to replication, involve enzymes that carry out their functions through the catalysis of metastable fuels into waste products. Catalytic control of reaction rates will prove to be as useful and ubiquitous in DNA nanotechnology as it is in biology. Here we present experimental results on the control of the decay rates of a metastable DNA "fuel". We show that the fuel complex can be induced to decay with a rate about 1600 times faster than it would decay spontaneously. The original DNA hybridization catalyst [15] achieved a maximal speed-up of roughly 30. The fuel complex discussed here can therefore serve as the basic ingredient for an improved DNA hybridization catalyst. As an example application for DNA hybridization catalysts, we propose a method for implementing arbitrary digital logic circuits.

1 Introduction

DNA has proven to be a highly versatile material for building artificial nanoscale devices. Among the devices already realized experimentally are DNA motors [6, 19, 7], DNA walkers [12, 13], DNA fuels [15], DNA catalysts [15] and self-assembled two dimensional crystals [16]. It is interesting to ask to what degree DNA alone can reproduce the richness of molecular biology and whether an alternative "DNA-only" world is conceivable. In fact, proposals that the history of life must include a time when nearly all the functions of life were subserved by RNA – the RNA World – have been given serious attention and are now widely accepted [3].

Biological systems exhibit complex and programmed behaviors. These behaviors are encoded by sets of specifically interacting molecules, such as DNA and proteins. The interactions among the molecules in such biochemical networks are akin to wires in electronic circuits. If we set our eyes on creating an artificial "cell" containing only DNA-based structures we need to design similar DNA-based biochemical networks that allow the components of the artificial cellular machinery to interact and communicate.

Biochemical circuits in biological systems almost universally rely on enzyme activity to carry out their function. It thus seems natural to use DNA catalytic systems as basic building blocks for a synthetic DNA-based circuit. While ribozymes are the best known example of nucleic acids with catalytic activity (such as DNA-cleaving or RNA-cleaving DNA enzymes [11, 1, 14]), we here want to use an alternative and entirely different type of DNA based catalytic system, namely a DNA hybridization catalyst [15]. DNA hybridization catalysts are rationally designed molecules that catalyze the conformational rearrangement of other DNA complexes. The catalytic system typically consists of two components: (i) a metastable "fuel", i.e. a DNA complex forced into a state from which it can not decay spontaneously into its true minimal energy configuration ("waste product"), and (ii) the actual catalyst which makes a fast pathway available for the transformation of the fuel complex into waste. The term fuel is used here, since the free energy that becomes available when the metastable complex decays can in principle be used to perform mechanical work. The fuel complex could for example serve as an energy source for a autonomous DNA-based molecular motor.

The first example of a DNA hybridization catalytic system was devised by Turberfield *et al.* [15]. There, the metastable fuel consisted of a pair of complementary strands of DNA that were inhibited from hybridization by inducing at least one of the strands to acquire a loop-like structure through partial hybridization with another strand of DNA. The inhibition was thought to be due to the difficulty a complementary strand of DNA has in threading its way through the loop to form duplex DNA. Catalytic speed-up was achieved by a short strand of DNA that binds on one side of the loop via a toehold and opens the loop through three-strand branch migration.

In Sec. 2 we describe this system in detail and propose modifications which promise to significantly enhance catalytic activity and applicability. In Sec. 3 we propose a scheme for linking such DNA hybridization catalysts into networks capable of performing complex logic. Finally, in Sec. 4 we will present experimental results on the construction and characterization of a DNA fuel complex and the catalytic control over the decay of the fuel complex.

2 Improving the DNA Hybridization Catalyst

Catalytic control of chemical reactions is essential for the design of autonomous behavior in biochemical systems. Two features are necessary for high performance of a catalytic system: programmability of the interactions so that catalyst systems can be tailored to a given task, and high ratios between the rates of catalyzed and uncatalyzed reactions. We expect that the former criterion will pose little difficulty for DNA hybridization catalysts, since catalyst design should be relatively insensitive to the specific choice of nucleotide sequence, once essential constraints are incorporated into the design process. The more interesting problem is to design a system with a high catalysis ratio.

Previous work has identified systematic approaches to the control of DNA hybridization kinetics. A key phenomenon, studied in [10, 4, 17, 18], is strand displacement by branch migration (Fig. 1a). Here, the reaction $P\bar{S} + S \rightarrow S\bar{S} + P$ is thermodynamically favorable, because of the additional base pairs that are formed. The kinetics of the reaction depends upon the length of the single-stranded overhang, known as a *toehold*. The toehold is where S initially binds to $P\bar{S}$, and the longer it is, the more likely that the reaction will enter a *branch*

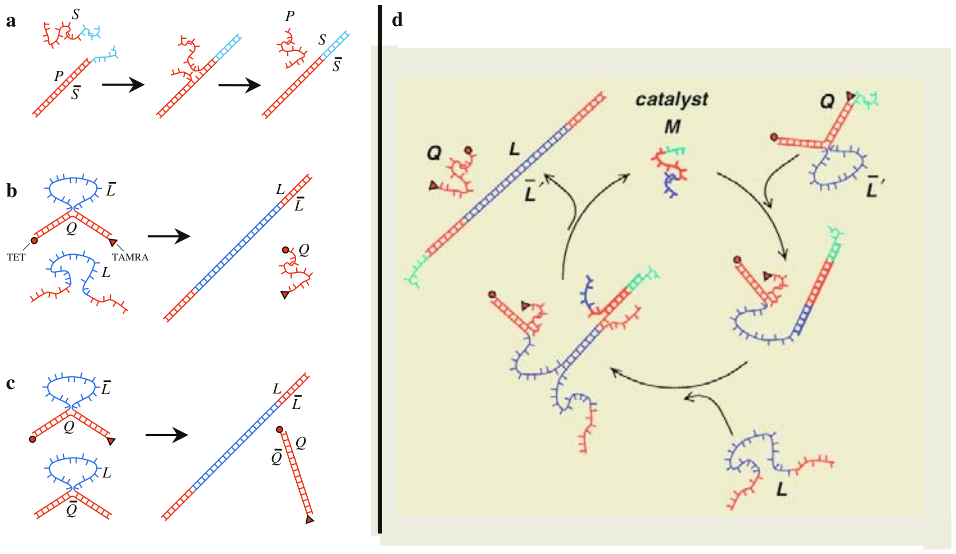


Fig. 1. **(a)** Single-stranded toehold allows rapid displacement of P by S. **(b)** The $Q\bar{L} + L \rightarrow Q + L\bar{L}$ reaction is slow. **(c)** The $Q\bar{L} + L\bar{Q} \rightarrow Q\bar{Q} + L\bar{L}$ reaction is even slower. **(d)** Strand M catalyzes the $Q\bar{L} + L \rightarrow Q + L\bar{L}$ reaction

migration phase prior to dissociation. Branch migration consists of isoenergetic steps where the final base pair of P to $\bar{S}$ is replaced by a base pairing of S to $\bar{S}$, thus moving the branch point by a random walk process [9, 8]. When the branch point reaches the left side of the complex, strand P dissociates. This is an essentially irreversible step, because there is no toehold for P; although, due to spurious "toeholds" produced by DNA breathing, the reaction can occur at low rates. Using fluorescence of a fluorophore/quencher pair to read out the bulk fraction of molecules in which the fluorophore and quencher were near each other (as in the random-coil state of Q), Yurke *et al.* [18] measured an exponential acceleration of reaction kinetics due to toehold lengths from 0 (where the rate constant is ~ 1 /M/s) to 6 (where the rate approaches that of ordinary hybridization, namely $\sim 10^6$ /M/s). The principle of strand displacement mediated by toeholds will be essential also for the more complicated systems considered below.

We now turn to DNA hybridization catalysts [15], where the reaction rate is controlled not by a permanent structural change in the molecules (such as adding a toehold or changing the length of a toehold) but rather by an additional strand, namely the *catalyst strand*. A precondition for such a system is a *metastable* DNA complex or pair of complexes for which a thermodynamically favorable but kinetically inaccessible configuration exists. A catalyst for this reaction must make available a fast pathway to the thermodynamically favorable state. Thus, the metastable complexes represent an energy source that can in principle be coupled to other reactions, with timing controlled by the catalyst.

An example is shown in Fig. 1b. As described in Ref. [15], the 40 nucleotide loop region was presumed to be tightly coiled, preventing hybridization in that domain. The second-order rate constant for this reaction was measured to be ~ 420 /M/s, which is $\sim 10^4$ times slower than the rate for the hybridization of unconstrained single-stranded DNA, such as L and $\bar{L}$. A catalyst for this reaction (Fig. 1d) was proposed and demonstrated in Ref. [15], where a 30-fold speed-up due to the catalyst strand M was measured.

To increase the catalysis ratio, however, we need a pair of metastable fuel complexes that have an even lower spontaneous hybridization rate. An obvious candidate is the system consisting of the molecules $Q\bar{L}$ and $L\bar{Q}$ shown in Fig. 1c where now both long strands L and $\bar{L}$ are forced into a loop. A preliminary study reported in Ref. [15] established an upper limit of 3/M/s for the second order rate constant of this reaction. The similarity with the singly protected system (Fig. 1b) suggests that catalysis of the hybridization reaction $Q\bar{L}+L\bar{Q} \rightarrow Q\bar{Q} + L\bar{L}$ will be possible.

Before turning to the experimental implementation (see Sec. 4) we will discuss one potential pathway for a catalyzed reaction between two loops (see Fig. 2) whose logic is based on the conclusions of [15]: The presumed compact state of the loop prevents interaction with single strands or other loops. To enable the loop-loop hybridization reaction, both of the fuel complexes must be opened, exposing both loop regions. This is accomplished by one molecule of the catalyst strand binding to each fuel complex. Thus, in the second hybridization step, both loop regions are open and available for hybridization. Once the combined complex has been formed, 3-way branch migration brings the conformation to the point where 4-way branch migration of the C arms may take place. After

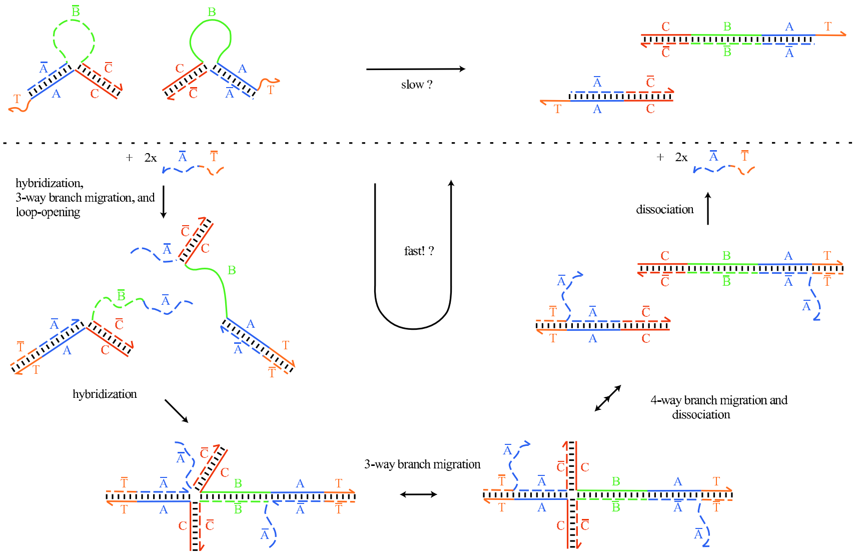

slow ?
+ 2x
hybridization, 3-way branch migration, and loop-opening
fast! ?
dissociation
hybridization
4-way branch migration and dissociation
3-way branch migration

Fig. 2. Two copies of strand M catalyze the $Q\bar{L} + L\bar{Q} \rightarrow Q\bar{Q} + L\bar{L}$ reaction of Fig. 1c. Here, subsequences of each strand are explicitly labeled, and the necessary toeholds are added. Thus, $M = \bar{T}\bar{A}$, $Q = CAT$, $\bar{L} = \bar{A}\bar{B}\bar{C}$, $\bar{Q} = \bar{A}\bar{C}$, and $L = CBAT$. The dotted line separates uncatalyzed reactions from those that occur only in the presence of catalyst strand $\bar{T}\bar{A}$

this final step, the complex dissociates into two inert waste products, from which the catalyst strands can dissociate due to the weak binding in the short toehold domain.

Unlike the original catalyst, which made use of 3-way branch migration only, the improved catalyst makes use of both 3-way branch migration (where individual steps are typically $\sim 10\ \mu$s) as well as 4-way branch migration (in which individual steps are typically ~ 100 ms depending upon reaction conditions) [9, 8]. Short 4-way branch migration reactions are typically completed in a few seconds, so we don't expect this to be the rate-limiting step. However, in another study we have discovered that initiation of branch migration at a junction can be surprisingly slow [20]. The results of that study suggest that a three nucleotide toehold would significantly enhance the reaction rates. To achieve this, we truncate the terminal three nucleotides of the catalyst $\bar{T}\bar{A}$, leaving 3nt of A unpaired to serve as a toehold to initiate 3-way branch migration.

Note that the reaction pathway shown in Fig. 2 is by no means the only one possible. In fact, the analysis of complex DNA hybridization pathways such as this one poses profound challenges, because the number of possible intermediate complex conformations (as measured by secondary structure) can be exceedingly large. For example, branch migration reactions in different parts of the molecule can occur at different rates and complete in different orders. Furthermore, what are shown as unstructured single-stranded regions may fold into weakly-structured states that can significantly affect the rates of reactions. Also, entirely unexpected reactions can in principle occur. One approach to these issues is to use stochastic models of secondary structure kinetics [2] to identify unexpected pathways or steps at which the reaction stalls. Initial simulations have demonstrated that the pathway shown here can in fact go to completion with sequences based on the original catalyst system.

3 Logic Gates and Circuits

In biological organisms, systems of catalysts regulate fundamental biological pathways. The ability of catalysts to regulate the activity of other catalysts allows complex logic to be implemented. The potential programmability of DNA hybridization catalysts may be ideal for the construction of logic circuits. However, for this to be accomplished, the catalysis of one pair of fuel complexes must be somehow coupled to the catalysis of a second pair of fuel complexes which have unrelated sequences. Our proposal for accomplishing this builds on the improved catalyst design by modifying one of the fuel complexes to contain an additional strand, $\bar{X}$, hybridized to the inside of the loop, as shown in Fig. 3. This strand is then released by strand displacement during an intermediate 3-way branch migration step in the reaction. Since the sequence of strand $\bar{X}$ is unrelated to the other parts (A, B, and C) of the catalyst design, we are free to choose its sequence to be the catalyst for an otherwise unrelated downstream hybridization reaction. By making the two toehold sequences distinct, a *different* catalyst strand is required to open each of the fuel complexes, and only presence of *both* of the input strands allows the reaction to go to comple-

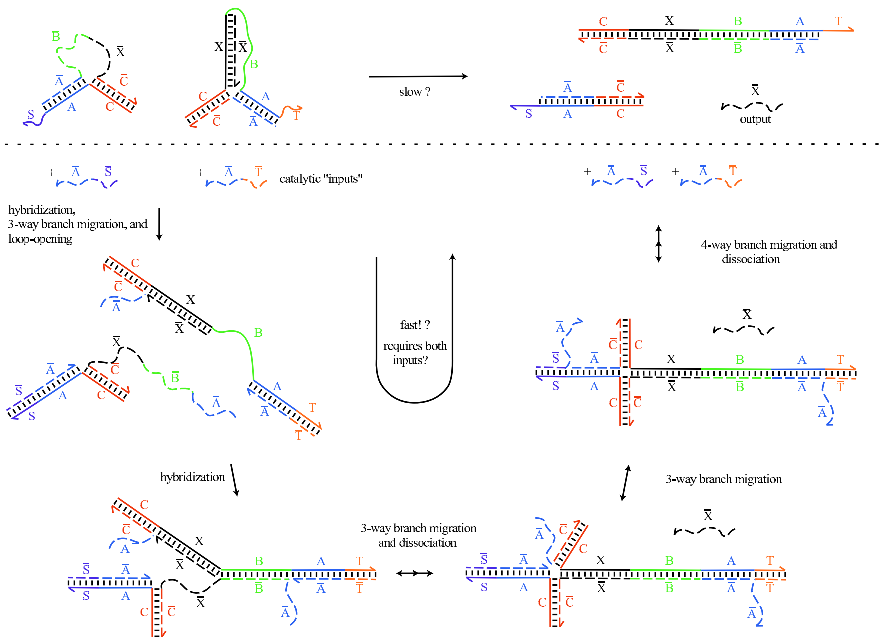

slow ?
output
catalytic "inputs"
hybridization, 3-way branch migration, and loop-opening
fast! ? requires both inputs?
4-way branch migration and dissociation
hybridization
3-way branch migration and dissociation
3-way branch migration

Fig. 3. An AND-gate catalyst constructed via two modifications of the improved catalyst shown in Fig. 2: (1) two distinct toehold sequences S and T guard access to the two loop-opening reactions required in the catalysis pathway; and (2) one of the loops contains a hybridized "output" strand, $\bar{X}$, which is released by strand displacement during the formation of $L\bar{L}$. Thus, the overall reaction can be written as $Gate(\bar{T}\bar{A}\ \&\ \bar{S}\bar{A} \Rightarrow \bar{X})$, indicating that $\bar{X}$ is produced as a single-stranded species only if both $\bar{T}\bar{A}$ and $\bar{S}\bar{A}$ are present. The dotted line separates uncatalyzed reactions from those that occur only in the presence of input strands

tion. This effects AND-gate logic, i.e., the output $\bar{X}$ is produced if and only if both $\bar{T}\bar{A}$ and $\bar{S}\bar{A}$ are present, which we write as $Gate(\bar{T}\bar{A}\ \&\ \bar{S}\bar{A} \Rightarrow \bar{X})$.

A simple variation of this gate results in OR-gate logic: give the toeholds from the A stems of both loops the same sequence T, and create analogous toeholds on both C stems with sequence S. Now, *either* input $\bar{T}\bar{A}$ or input $\bar{S}\bar{C}$ will be sufficient to trigger the catalytic step and release the output strand $\bar{X}$. We write this as $Gate(\bar{T}\bar{A} \mid \bar{S}\bar{C} \Rightarrow \bar{X})$.

To implement any desired computation, a universal gate such as NAND is needed. AND and OR provide a universal basis only for monotone circuits, which can directly implement a strict subclass of all boolean functions, specifically those for which an input bit flipping from 0 to 1 can never cause the output to flip from 1 to 0. However, this seeming limitation is lifted if we allow a "dual rail" input representation, wherein distinct signals x_i^+ and x_i^- are used to represent "$x_i = 0$" and "$x_i = 1$". Either $x_i^+ = 1$ or $x_i^- = 1$, but not both; however, both x_i^+ and x_i^- can be 0, indicating that bit x_i has not yet been computed. To see that an arbitrary function $x_n = f(x_1, x_2, ...x_m)$ can be implemented with monotone gates, consider a circuit of NAND gates that implements $f(\cdot)$. Replace each gate $x_k = x_i$ NAND x_j by the pair of monotone gates $x_k^+ = x_i^-$ OR x_j^- and $x_k^- = x_i^+$ AND x_j^+. Computation in the new circuit begins with all variable at zero, then the appropriate input variables are flipped to 1 and subsequent downstream gates are evaluated. (Note: if a gate is evaluated prematurely, and both inputs to an OR or AND gate are 0, nothing happens to the output either.) As soon as the signal reaches the output gate n, either x_n^+ or x_n^- flips to 1, giving the result of the computation.

It is straightforward to use DNA hybridization AND-gates and OR-gates to implement arbitrary digital circuits using this "dual-rail" monotone logic. Indeed, because we have proposed no mechanism for getting rid of a catalyst once it is released, the monotone logic property is essential for the circuits we could build this way. To further minimize the size of constructed circuits, and to show that the shared sequence requirement of the AND-gate catalyst's inputs causes no difficulties, we use two optimizations. First, all OR gates may be eliminated and replaced by a "wired-OR"; i.e., each gate that outputs one of the OR gate's inputs can now instead directly produce the OR gate's output itself. The resulting circuit can be pictured as in Fig. 4. The inputs to the new AND gate i are distinguished as type "T" or type "S", corresponding to whether the DNA input strands will be $\bar{T}_i\bar{A}_i$ or $\bar{S}_i\bar{A}_i$ respectively, where the subscript i indicated that distinct DNA sequences will be used for each gate. Now, for each distinct output wire from each AND gate, we construct an AND-gate catalyst with the given inputs and the given output for that wire. Noting that arbitrary fan-out can be achieved, this completes the construction.

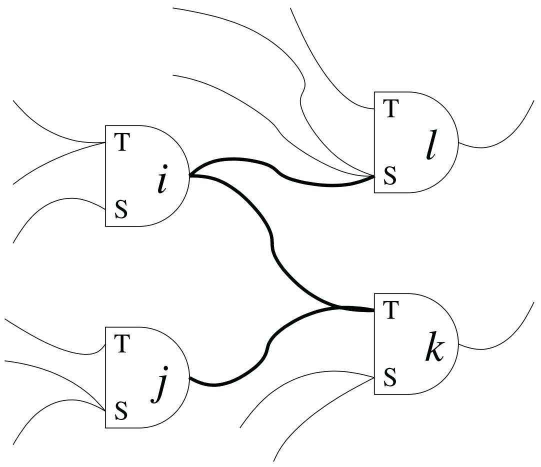


Fig. 4. A monotone logic circuit with AND gates and wired OR. The two inputs to each AND gate are distinguished. In this diagram, if either gate i or gate j outputs 1, the "T" input of gate k is activated. To translate this circuit to a DNA hybridization catalyst network, we require one AND-gate catalyst for each wire in the diagram. For example, the three thick wires become the three catalyst gates $Gate(\bar{T}_i\bar{A}_i \ \& \ \bar{S}_i\bar{A}_i \Rightarrow \bar{S}_l\bar{A}_l)$, $Gate(\bar{T}_i\bar{A}_i \ \& \ \bar{S}_i\bar{A}_i \Rightarrow \bar{T}_k\bar{A}_k)$, $Gate(\bar{T}_j\bar{A}_j \ \& \ \bar{S}_j\bar{A}_j \Rightarrow \bar{T}_k\bar{A}_k)$

In a large circuit consisting of DNA hybridization catalyst gates, input is provided by adding the catalysts corresponding to 1 or 0 input variables, as appropriate. This will trigger the release of the logically correct intermediate catalysts for intermediate gates, and finally, the release of either the 1 output strand or the 0 output strand. Recognizing that the uncatalyzed rate of each hybridization reaction is non-zero, we must consider that eventually every intermediate and every output catalyst species will be released. Thus, to evaluate the probable reliability of circuits constructed this way, understanding the timing of reactions is essential: will the 1 output strand or the 0 output strand be produced *first*? It is critical, for example, that the presence of a single catalyst input to an AND gate will not significantly accelerate the reaction. In some cases a biochemical AND gate, with both inputs driven by the same species, can function as a restoring element to correct small errors [5]; it remains to be seen to what extent the sigmoidal input/output curve expected for the AND-gate catalyst can be used to reduce errors in this context.

It is remarkable that active biochemical logic can in principle be accomplished by DNA without the assistance of any enzymes. Note however that we have only shown how to construct *feed forward* circuits; it is an open question whether the general scheme presented here can be adapted to the case of feedback circuits, where signals can be turned off and on dynamically.

4 Experimental Implementation

From the discussion in the previous section it is clear that the experimental realization of a DNA-based catalyst with a large catalysis ratio is a necessary precondition for the implementation of a DNA-logic circuit. Our preliminary experimental results indicate that the system outlined in Sec. 2 may indeed work as a catalyst with a remarkably large catalysis ratio. However, the experiment also shows several interesting deviations from the theoretical scheme.

Typical experimental data for the reaction between the two loops $Q\bar{L}$ and $L\bar{Q}$ (in the following we will use the notation introduced in Fig. 1) is shown in Fig. 5. In this experiment, the strands $\bar{L}$ and Q are unlabeled, strand $\bar{Q}$ is fluorescence-labeled with the fluorophore TAMRA at the $3'$-end and L is labeled with an Iowa Black quencher at the $5'$-end. Thus, the reactant $L\bar{Q}$ is dark, while the product $Q\bar{Q}$ is fluorescent. The loops are formed in a slow anneal. The experiment shown was performed at a constant temperature $T = 20°\text{C}$ and with a concentration $c = 0.5\ \mu\text{M}$ for the two loops. The reaction is initiated when the $Q\bar{L}$-solution is added to the $L\bar{Q}$-solution. An increase in fluorescence intensity is a direct measure of the rate of dissolution of the $L\bar{Q}$-loop as the reaction $Q\bar{L} + L\bar{Q} \rightarrow Q\bar{Q} + L\bar{L}$ proceeds. In the example shown in Fig. 5 we followed the reaction over more than three days and, as can be seen from the figure, the fluorescence intensity changes very little after the first few hours or so. One might thus conclude that after this point almost all the available loops have reacted and only inert segments of double-stranded DNA are present in the solution. However, surprisingly, annealing the sample up to 80°C for 5 minutes at the end of the measurement and subsequently remeasuring the fluorescence intensity shows an increase in fluorescence of around 25 percent.

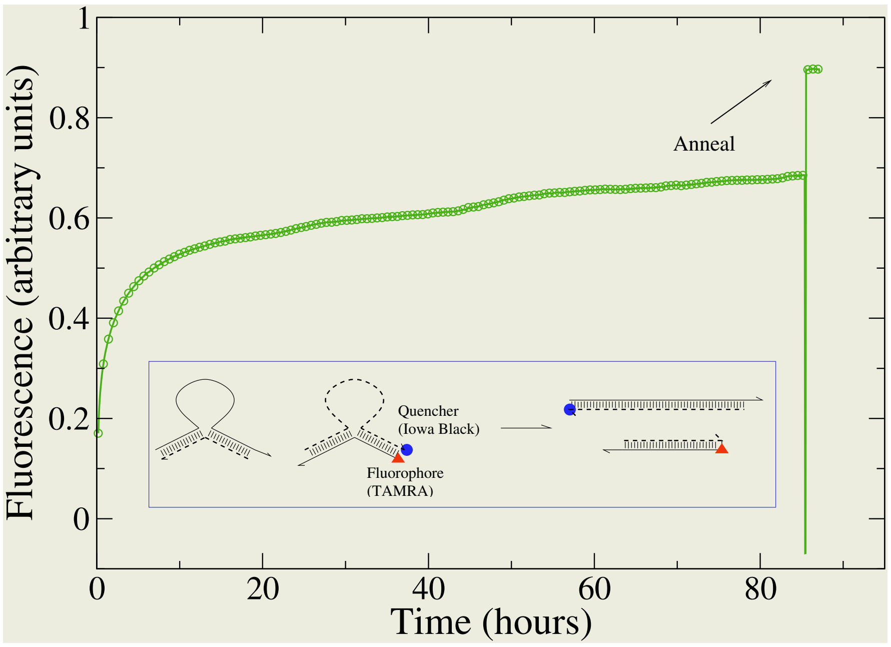


Fig. 5. The figure shows the long-time behavior of a stoichiometric mixture of the two loops $Q\bar{L}$ and $L\bar{Q}$. Strands $\bar{L}$ and Q are unlabeled, strand $\bar{Q}$ is fluorescence-labeled with the fluorophore TAMRA at the $3'$-end and L is labeled with an Iowa Black quencher at the $5'$-end (see inset). In the initial state fluorescence is quenched. The increase in fluorescence intensity is a measure of the progress of the reaction $Q\bar{L} + L\bar{Q} \rightarrow Q\bar{Q} + L\bar{L}$. Remarkably, only the anneal performed after $\sim$ 86 hours brings the reaction to completion. This indicates the presence of a metastable compound in the solution. The experiment is performed in SPSC buffer (pH 6.5, 1 M NaCl) at a temperature $T = 20°$C

From this experiment we can draw two main conclusions: First, as the relatively rapid initial increase in fluorescence indicates, the mixture of the two loops $Q\bar{L}$ and $L\bar{Q}$ is less stable than one might have expected. In fact, a more detailed analysis shows that the reaction $Q\bar{L} + L\bar{Q} \rightarrow Q\bar{Q} + L\bar{L}$ initially progresses at a rate comparable to that of the reaction $\bar{L}Q + L \rightarrow Q + L\bar{L}$ where only one of the two long strands is protected (see Fig. 1b). Taken alone, this would indicate that the catalyst proposed in Sec. 2 is at best a minor improvement over the simpler system of Ref. [15]. Our second finding, however, is more intriguing: Even after several days the reaction has not gone to completion, as indicated by the large jump in fluorescence intensity upon annealing. What is more, an electrophoresis gel shows that directly before the anneal three species are present in the solution mixture (see Fig. 6). Two of them are the expected reaction end products $L\bar{L}$ and $Q\bar{Q}$ while the third one is a complex with a mobility roughly half that of a single loop. After the anneal, only $L\bar{L}$ and $Q\bar{Q}$ are found.

While it is difficult to determine the exact structure of the low-mobility species, we have verified that all four strands (L, $\bar{L}$, Q and $\bar{Q}$) used for the

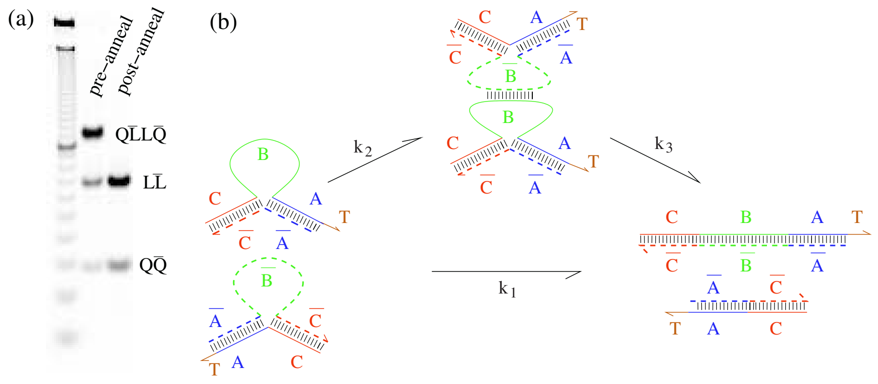


Fig. 6. (a) Products of the reaction between the two loops $Q\bar{L}$ and $L\bar{Q}$. The reaction was left to proceed for about twelve hours (T=20°C, TAE buffer, pH 8, 12.5 mM Mg^{++}) before the gel was run. Left lane: ladder with ten base pair spacing. Middle lane: the three bands correspond to the metastable fuel complex $Q\bar{L}L\bar{Q}$ (lowest mobility) and to the double stranded end products $L\bar{L}$ and $Q\bar{Q}$ (highest mobility). Right lane: After an anneal, only the stable end products $L\bar{L}$ and $Q\bar{Q}$ are found. No bands corresponding to unreacted loops are seen (the mobility of the unreacted loops is comparable to that of a 55*mer*). **(b)** Proposed reaction pathway: The two loops $Q\bar{L}$ and $L\bar{Q}$ can either decay directly into the end products $L\bar{L}$ and $Q\bar{Q}$ (rate constant k_1) or via a metastable complex. In our model, k_2 is the second order rate constant for the formation of the intermediate complex and k_3 is the first order rate constant describing its subsequent decay. Subsequences of each strand are explicitly labeled: $Q = CAT$, $\bar{L} = \bar{A}\bar{B}\bar{C}$, $\bar{Q} = \bar{A}\bar{C}$, and $L = CBAT$

formation of the loops are also present in this complex. It is therefore plausible that the complex is formed due to strong interactions between the single stranded regions of the two loops.

Two competing reactions appear to be taking place when solutions of the two loops are mixed. In one of them, the final double-stranded minimum-energy compounds are formed directly. Let the rate constant for this reaction be k_1. The second, parallel reaction proceeds via a metastable intermediate which only slowly decays into the end products $L\bar{L}$ and $Q\bar{Q}$. We model the rate constants for these two steps as k_2 and k_3, respectively (see Fig. 6). The putative reactions taking place can be summarized as follows:

$$Q\bar{L} + L\bar{Q} \overset{k_1}{\to} Q\bar{Q} + L\bar{L}, \tag{1}$$

$$Q\bar{L} + L\bar{Q} \overset{k_2}{\to} Q\bar{L}L\bar{Q} \overset{k_3}{\to} Q\bar{Q} + L\bar{L}. \tag{2}$$

To estimate the rate constants k_1, k_2 and k_3 we use a simple mathematical model for the reaction kinetics. The reaction given in Eq. 1 and the first step of the reaction in Eq. 2 are assumed to be second order, while the second step of the reaction in Eq. 2 is treated as a first order reaction. The fluorescence concentration is assumed to be proportional to the concentration of $Q\bar{Q}$ and it is further assumed that the fluorescence remains quenched in the complex

$Q\bar{L}L\bar{Q}$. From the slow decay of the fluorescence intensity at long times it is clear that the rate for the decay of $Q\bar{L}L\bar{Q}$ must be small.

The metastable complex $Q\bar{L}L\bar{Q}$ is dissolved very easily upon addition of the catalyst strand M as is shown in Fig. 7. The loops used in this reaction are dye-labeled as described above and mixed stoichiometrically at a concentration $c = 0.5\ \mu$M. Catalyst strand M at a concentration of $c = 0.3\ \mu$M was added after about 12 hours. The reaction then very rapidly went to completion. In fact, a final anneal did not markedly change the fluorescence intensity in contrast to our previous result.

To estimate the reaction speed up we need to compare the rate constants before and after catalyst addition. The uncatalyzed reaction is modeled as outlined above (see Fig. 6 and Eqs. 1 and 2): The data in Fig. 7 is well approximated by our model with $k_1 = 690$ /M/s, $k_2 = 670$ /M/s and $k_3 = 10^{-6}$ /s (see Fig. 7). The values for the rate constants given here should be considered as order of magnitude estimates. As we have already mentioned previously, the rate constants k_1 and k_2 are of the same order of magnitude as the rate constant for

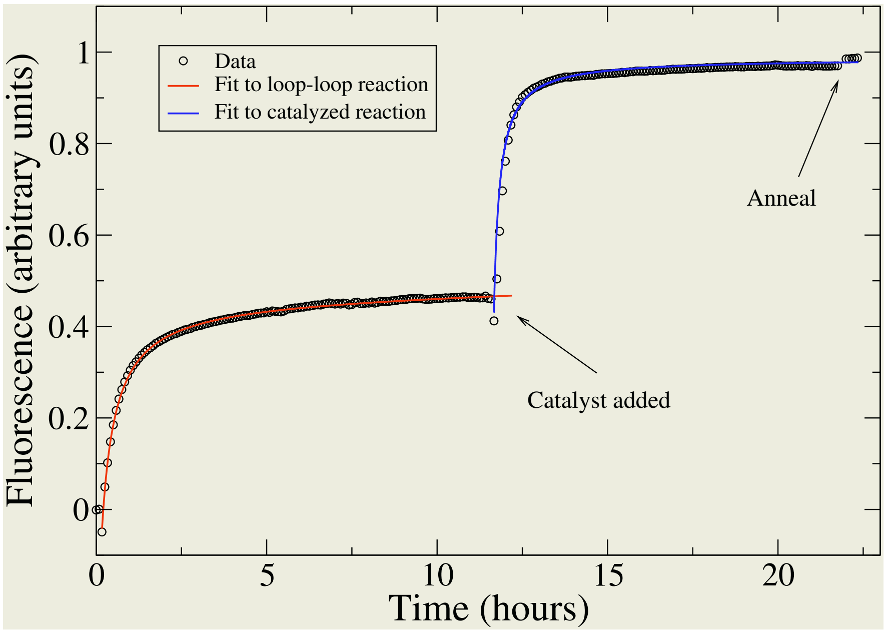
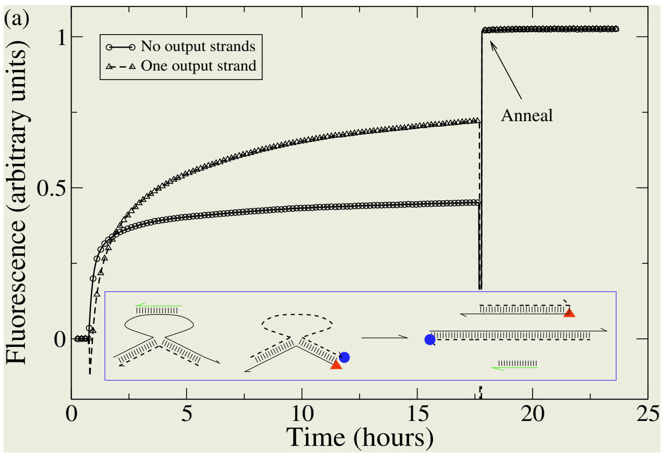


Fig. 7. Catalysis of the reaction between the two loops $Q\bar{L}$ and $L\bar{Q}$. The experiment was performed in TAE buffer (pH 8, 12.5 mM Mg^{++}) at a temperature $T = 20°$C with an initial concentration $c = 0.5\ \mu$M for both loops. The locations of the dye label and quencher are as indicated in the inset of Fig. 5. Catalyst strand M (concentration $c = 0.3\ \mu$M) is added after approximately 12 hours. After the addition of the catalyst the reaction goes to completion and a final anneal increases the fluorescence only marginally. The full lines are fits to the data using the model described in the text. For the rate constants we use the numerical values $k_1 = 690$ /M/s, $k_2 = 670$ /M/s, $k_3 = 10^{-6}$ /s and $k_4 = 5300$ /M/s

the reaction $L\bar{Q} + \bar{L} \rightarrow \bar{Q} + L\bar{L}$ (see Ref. [15]). The reaction after addition of catalyst strand can be fitted with a simple second order rate law with a rate constant $k_4 = 5.3 \times 10^3$ /M/s (see Fig. 7). Since the catalyst concentration is $c = 0.3$ μM the half-time $\tilde{t}_{1/2}$ for this reaction is $1/k_4c = 625$ s. The half time $t_{1/2}$ for the uncatalyzed decay of the metastable compound, on the other hand, is $t_{1/2} = 1/k_3 = 10^6$ s. If we define the reaction speed up as the ratio of the half times for the decay of the metastable compound before and after addition of catalyst strand we obtain $t_{1/2}/\tilde{t}_{1/2} = 1600$.

In Sec. 3 we proposed a modification of the original dual catalyst design where an additional strand $\bar{X}$ is hybridized to the inside of the loop. This strand is released when the loops react and can itself act as a catalyst in a downstream reaction. The proposal for implementing logic gates outlined in Sec. 3 relied on two assumptions: (i) that the two loops remain stable and independent in solution and (ii) that there is a clear separation of timescales between the reactions taking place in the absence and presence of input strands. The surprising existence of the intermediate $Q\bar{L}L\bar{Q}$ foreshadows that the scheme discussed in Sec. 3 will have to be modified at least in part.

Intuitively, one would expect strand $\bar{X}$ to hinder loop-loop interactions and thus to suppress complex formation. That this is indeed the case can be seen from the preliminary experimental data shown in Fig. 8. There, the reaction between the modified loop $L\bar{Q}\bar{X}$ and loop $Q\bar{L}$ is compared to the reaction between $L\bar{Q}$ and $Q\bar{L}$. In our experiment, the output strand $\bar{X}$ has a length of 21 bases (the same length as the catalyst strand) and its sequence is complementary to the



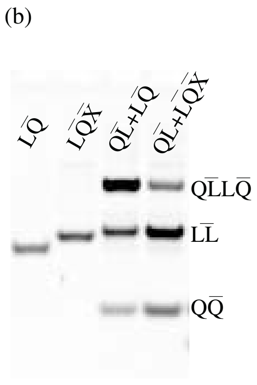


Fig. 8. (a) Comparison of the reactions between the two unmodified loops $Q\bar{L}$ and $L\bar{Q}$ and the reaction between $Q\bar{L}$ and $L\bar{Q}\bar{X}$. The experiment was performed in TAE buffer (pH 8, 12.5 mM Mg^{++}) at a temperature $T = 20°$C with an initial reactant concentration $c = 0.5$ μM. The locations of the dye label, quencher and output strand $\bar{X}$ are as indicated in the inset. **(b)** Reactants and reaction products for the reaction between $Q\bar{L}$ and $L\bar{Q}\bar{X}$. Far left lane: Loop $Q\bar{L}$. Center left lane: Modified loop $L\bar{Q}\bar{X}$. Center right lane: Products of the reaction between $Q\bar{L}$ and $L\bar{Q}$. Far right lane: Products of the reaction between $Q\bar{L}$ and $L\bar{Q}\bar{X}$

central part of L. To form the modified loop $L\bar{Q}\bar{X}$ the output strand is added to a solution of preformed loops $L\bar{Q}$. The left two lanes in the gel of Fig. 8b show the band corresponding to the unmodified loop $Q\bar{L}$ (far left) and to the (less mobile) modified loop $L\bar{Q}\bar{X}$ (center left).

The fluorescence data (Fig. 8a) shows that the reaction with one modified loop proceeds slower than that between the unmodified loops but also goes closer to completion. An explanation for the latter is provided by the gel in Fig. 8b, center right and far right lanes, which shows that in the presence of output strands relatively more loops participate in a reaction that leads to end product formation than to complex formation. Contrary to our assumptions, even in the modified reaction essentially all the loops seem either to decay into double-stranded end products or to form metastable complexes and do not remain in solution in their original form. While no band corresponding to free output strand $\bar{X}$ is seen in the gel, it seems plausible that $\bar{X}$ is released in both the decay and the complex formation reaction. (This is confirmed indirectly since the bands corresponding to all reaction products migrate at the same speed in both cases.)

5 Discussion

The slow long-term increase in fluorescence intensity we observed for $Q\bar{L}+L\bar{Q}$ is in qualitative agreement with the preliminary results on loop-loop reactions reported in [15]. A quantitative comparison is difficult since reactant concentration, location of dye and quencher within the molecules, as well as the dye-quencher pair used, differ between our work and the work reported in [15]. However, our results shed light on the actual origin of the observed stability. The discovery that $Q\bar{L}$ and $L\bar{Q}$ form a stable complex also leads to the view that the reaction depicted in Fig. 1b proceeds through the formation of a bound complex between $Q\bar{L}$ and L that then slowly decays to $L\bar{L}$ and Q. This differs from the intuitive view, expressed earlier, that the reaction is inhibited because of tight coiling of the loop. Furthermore, it now seems probable that the catalysis cycle shown in Fig. 1d proceeds by a different pathway (for substoichiometric catalyst strand concentrations): $Q\bar{L}$ first forms a complex with L, then the catalyst strand accelerates its decay into Q and $L\bar{L}$.

Further work is needed to demonstrate catalytic speed-up of the reaction leading to the decay of the metastable compound. So far, we have only measured the reaction speed-up in the case where catalyst strand and complex are mixed roughly stoichiometrically. The result thus obtained can be considered an estimate for an upper bound on the reaction rates. However, to show that the catalyst strand indeed acts catalytically, it will be necessary to demonstrate turnover – *i.e.* to show that the reaction goes to completion for sub-stoichiometric amounts of catalyst strand and that in this regime the reaction rate increases (linearly) with catalyst concentration.

To implement an OR gate or an AND gate one would have to demonstrate that adding input strands to a mixture of $L\bar{Q}\bar{X}$ and $Q\bar{L}$ speeds up the release of the output strand $\bar{X}$. However, contrary to the situation encountered in the catalysis experiments discussed above, it is fruitless to add an input strand to the solution after the metastable complex has formed, because presumably all output strands have already been released at this point. Alternatively, catalyst strands could be added when the two loops are mixed initially, and their effectiveness measured by a speed-up in the release of the output strands. For such an experiment it seems most convenient to choose an intramolecular dye/quencher configuration that allows one to directly monitor the release of the output strand. It seems probable that at least the relatively simpler OR gate can be implemented according to the scheme outlined here, as all it requires is the speed-up of the initial reaction between $L\bar{Q}\bar{X}$ and $Q\bar{L}$ in the presence of either of two input strands. The on/off ratio for such a gate can be expected to be comparable to the ratio between the catalyzed and uncatalyzed reactions of Ref. [15] which was found to be about 20 to 30. However, for implementing more complicated circuits in which several gates are linked, it seems more promising to completely redesign the constituent gates using the metastable complex explicitly as a starting point. In this way, one could hope to design gates with an on/off ratio of the order of one thousand.

Acknowledgments. Thanks to Ben Rahn, Jeremy Leibs, Joseph Schaeffer, Jongmin Kim, Dave Zhang, and especially Paul Rothemund for stimulating discussion and help preparing figures and simulations. GS was supported by the Swiss National Science Foundation, EW was supported by NSF CAREER Grant No. 0093486, NSF ITR Grant No. 0113443, and GenTel.

References

1. N. Carmi, S. R. Balkhi, and R. R. Breaker. Cleaving DNA with DNA. *Proceedings of the National Academy of Sciences*, 95:2233–2237, 1998.
2. C. Flamm, W. Fontana, I. Hofacker, and P. Schuster. RNA folding at elementary step resolution. *RNA*, 6:325–338, 2000.
3. R. F. Gesteland, T. R. Cech, and J. F. Atkins. *The RNA world.* Cold Spring Harbor Laboratory Press, Cold Spring Harbor, New York, 1999.
4. C. Green and C. Tibbetts. Reassociation rate-limited displacement of DNA strands by branch migration. *Nucleic Acids Research*, 9:1905–1918, 1981.
5. M. O. Magnasco. Chemical kinetics is Turing universal. *Physical Review Letters*, 78(6):1190–1193, 1997.
6. C. D. Mao, W. Q. Sun, Z. Y. Shen, and N. C. Seeman. A nanomechanical device based on the B-Z transition of DNA. *Nature*, 397:144–146, Jan. 14, 1999.
7. C. M. Niemeyer and M. Adler. Nanomechanical devices based on DNA. *Angewandte Chemie International Edition*, 41(20):3779–3783, 2002.
8. I. G. Panyutin, I. Biswas, and P. Hsieh. A pivotal role for the structure of the Holliday junction in DNA branch migration. *The EMBO Journal*, 14(8):1819–1826, 1995.
9. I. G. Panyutin and P. Hsieh. Kinetics of spontaneous DNA branch migration. *Proceedings of the National Academy of Sciences*, 91:2021–2025, 1994.
10. C. Radding, K. Beattie, W. Holloman, and R. Wiegand. Uptake of homologous single-stranded fragments by superhelical DNA. IV. branch migration. *J. Mol. Biol.*, 166:825–839, 1977.
11. S. W. Santoro and G. F. Joyce. A general purpose DNA cleaving RNA enzyme. *Proceedings of the National Academy of Sciences USA*, 94:4262–4266, 1997.

12. W. B. Sherman and N. C. Seeman. A precisely controlled DNA biped walking device. *Nano Letters*, 4(7):1203–1207, 2004.
13. J. Shin and N. Pierce. A synthetic DNA walker for molecular transport. *Journal of the American Chemical Society*, 126(35):10834–10835, 2004.
14. M. N. Stojanovic, T. E. Mitchell, and D. Stefanovic. Deoxyribozyme-based logic gates. *Journal of the American Chemical Society*, 124:3555–3561, 2002.
15. A. J. Turberfield, J. C. Mitchell, B. Yurke, A. P. Mills, Jr., M. I. Blakey, and F. C. Simmel. DNA fuel for free-running nanomachines. *Physical Review Letters*, 90(11):118102–1–4, 2003.
16. E. Winfree, F. Liu, L. A. Wenzler, and N. C. Seeman. Design and self-assembly of two-dimensional DNA crystals. *Nature*, 394:539–544, 1998.
17. D. M. Wong, P. H. Weinstock, and J. G. Wetmur. Branch capture reactions: displacers derived from asymmetric PCR. *Nucleic Acids Research*, 19:2251–2259, 1991.
18. B. Yurke and A. P. Mills, Jr. Using DNA to power nanostructures. *Genetic Programming and Evolvable Machines*, 4:111–122, 2003.
19. B. Yurke, A. J. Turberfield, A. P. Mills, Jr., F. C. Simmel, and J. L. Neumann. A DNA-fuelled molecular machine made of DNA. *Nature*, 406:605–608, 2000.
20. D. Y. Zhang and J. Schaeffer. Personal communication, 2003.

Complexity of Self-assembled Shapes

(Extended Abstract*)

David Soloveichik and Erik Winfree

California Institute of Technology, Pasadena, CA 91125, USA
{dsolov, winfree}@caltech.edu

Abstract. The connection between self-assembly and computation suggests that a shape can be considered the output of a self-assembly "program," a set of tiles that fit together to create a shape. It seems plausible that the size of the smallest self-assembly program that builds a shape and the shape's descriptional (Kolmogorov) complexity should be related. We show that under the notion of a shape that is independent of scale this is indeed so: in the Tile Assembly Model, the minimal number of distinct tile types necessary to self-assemble an arbitrarily scaled shape can be bounded both above and below in terms of the shape's Kolmogorov complexity. As part of the proof of the main result, we sketch a general method for converting a program outputting a shape as a list of locations into a set of tile types that self-assembles into a scaled up version of that shape. Our result implies, somewhat counter-intuitively, that self-assembly of a scaled up version of a shape often requires fewer tile types, and suggests that the independence of scale in self-assembly theory plays the same crucial role as the independence of running time in the theory of computability.

1 Introduction

Self-assembly is the process by which an organized structure can spontaneously form from simple parts. The Tile Assembly Model [15, 14], based on Wang tiling [13], formalizes the two-dimensional self-assembly of square units called "tiles" using a physically plausible abstraction of crystal growth. In this model, a new tile can adsorb to a growing complex if it binds strongly enough. Each of the four sides of a tile has an associated bond type that interacts with a certain strength with matching sides of other tiles. The process of self-assembly is initiated by a single seed tile and proceeds via the sequential addition of new tiles. Confirming the physical plausibility and relevance of the abstraction, simple self-assembling systems of tiles have been built out of certain types of DNA molecules [16, 11, 10, 8]. The possibility of using self-assembly for nanofabrication of complex components such as circuits has been suggested as a promising application [5].

* A preprint of the full paper can be found at http://arxiv.org.

The view that the "shape" of a self-assembled complex can be considered the output of a computational process [2] has inspired recent interest [7, 1, 3, 6, 4]. While it was shown through specific examples that self-assembly could be used to construct interesting shapes and patterns, it was not known in general which shapes could be self-assembled from a small number of tile types. Understanding the complexity of shapes is facilitated by an appropriate definition of shape. In our model, a tile system generates a particular shape if it produces any scaled version of that shape (Sect. 3). This definition may be thought to formalize the idea that a structure can be made up of arbitrarily small pieces. Computationally, it is analogous to disregarding computation time and is thus more appropriate as a notion of output of a *universal* computation process.[1] Using this definition of shape, we show that for any shape $\tilde{S}$, if $K_{sa}(\tilde{S})$ is the minimal number of distinct tile types necessary to self-assemble it then $K_{sa}(\tilde{S}) \log K_{sa}(\tilde{S})$ is within multiplicative and additive constants (independent of $\tilde{S}$) of the shape's Kolmogorov complexity. This theorem is proved by construction (which might be of independent interest) of a general method for converting a program that outputs a fixed size shape as a list of locations into a tile system that self-assembles a scaled version of the shape. Our result ties the computation of a shape and its self-assembly, and, somewhat counter-intuitively, implies that scaling up a shape may often allow it to be self-assembled from fewer tile types. Another consequence of the theorem is that the minimal number of tile types necessary to self-assemble an arbitrary scaling of a shape is uncomputable. Answering the same question about shapes of a fixed size is computable but NP complete [1].

2 The Tile Assembly Model

We present a description of the Tile Assembly Model based on Rothemund and Winfree [7] and Rothemund [6]. We will be working on a $\mathbb{Z} \times \mathbb{Z}$ grid of unit square locations. The **directions** $\mathcal{D} = \{N, E, W, S\}$ are used to indicate relative positions in the grid. Formally, they are functions $\mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z} \times \mathbb{Z}$: $N(i, j) = (i, j+1)$, $E(i, j) = (i+1, j)$, $S(i, j) = (i, j-1)$, and $W(i, j) = (i-1, j)$. The inverse directions are defined naturally: $N^{-1}(i, j) = S(i, j)$, etc. Let Σ be a set of **bond types**. A **tile type** $\boxed{t}$ is a 4-tuple $(\sigma_N, \sigma_E, \sigma_S, \sigma_W) \in \Sigma^4$ indicating the associated bond types on the north, east, south, and west sides. Note that tile types are oriented and a rotated version of a tile type is considered to be a different tile type. A special bond type *null* represents the lack of an interaction and the special tile type $\mathit{empty} = (\mathit{null}, \mathit{null}, \mathit{null}, \mathit{null})$ represents an empty space. If T is a set of tile types, a **tile** is a pair $(\boxed{t}, (i, j)) \in T \times \mathbb{Z}^2$ indicating that location (i, j) contains the tile type $\boxed{t}$. Given the tile $t = (\boxed{t}, (i, j))$, $\mathit{type}(t) = \boxed{t}$

[1] The production of a shape of a fixed size cannot be considered the output of a universal computation process: whereas questions about the result of universal computation are often uncomputable, any question about a shape of a fixed-size can be answered with a finite simulation of the self-assembly process [7], because in the model considered here, once a tile is added, it cannot be removed. Thus questions about shapes of fixed size are decidable.

and $pos(t) = (i, j)$. Further, $bond_D(\boxed{t})$, where $D \in \mathcal{D}$, is the bond type of the respective side of $\boxed{t}$, and $bond_D(t) = bond_D(type(t))$. A **configuration** is a set of tiles such that there is exactly one tile in every location $(i, j) \in \mathbb{Z} \times \mathbb{Z}$. For any configuration A, we write $A(i, j)$ to indicate the tile at location (i, j). We will write a configuration as a set of non-empty tiles; all other tiles are implicitly *empty*.

A **strength function** $g : \Sigma \times \Sigma \to \mathbb{Z}$, where $null \in \Sigma$, defines the interactions between adjacent tiles: we say that a tile t_1 interacts with its neighbor t_2 with strength $\Gamma(t_1, t_2) = g(\sigma, \sigma')$ where σ is the bond type of tile t_1 that is adjacent to the bond type σ' of tile t_2.[2] In most previous works on self-assembly, as in this work, strength functions are restricted with the following properties: (1) g is symmetric (the effect that one tile has an another is equal to the effect that the other has on the first), (2) $g(\sigma, null) = 0$ (the lack of an interaction is normalized to zero), (3) g is *non-negative* (there are no "adverse" interactions counteracting other interactions), (4) g is *diagonal*: $g(\sigma, \sigma') = 0$ if $\sigma \neq \sigma'$ (only sides with matching bond types interact). Property 4 shows the roots of the Tile Assembly Model in the Wang tiling model. Our results confirm that non-negativity is not a major restriction as Theorem 1 is valid for strength functions taking on negative values. However, if property 4 is relaxed then information in tile systems can be represented more compactly (using fewer tile types [4]), potentially leading to a different form of Theorem 1.

A **tile system T** is a quadruple (T, t_s, g, τ) where T is a finite set of non-empty tile types, t_s is a special **seed tile** with $type(t_s) \in T$, g is a strength function, and τ is the threshold parameter. Self-assembly is defined by a relation between configurations. Suppose A and B are two configurations, and t is a tile such that $A = B$ except at $pos(t)$ and $A(pos(t)) = null$ but $B(pos(t)) = t$. Then we write $A \to_{\mathbf{T}} B$ if $\sum_{D \in \mathcal{D}} \Gamma(t, A(D(pos(t)))) \geq \tau$. This means that a tile can be added to a configuration iff the sum of its interaction strengths with its neighbors reaches τ. The relation $\to^*_{\mathbf{T}}$ is the reflexive transitive closure of $\to_{\mathbf{T}}$.

Whereas a configuration can be any arrangement of tiles (not necessarily connected), we are interested in the subclass of configurations that can result from a self-assembly process. Formally, the tile system and the relation $\to^*_{\mathbf{T}}$ define the partially ordered set of **assemblies**: $Prod(\mathbf{T}) = \{A \text{ s.t. } \{t_s\} \to^*_{\mathbf{T}} A\}$, and the set of **terminal assemblies**: $Term(\mathbf{T}) = \{A \in Prod(\mathbf{T}) \text{ and } \nexists B \neq A \text{ s.t. } A \to^*_{\mathbf{T}} B\}$. A tile system **T uniquely produces** A if $\forall B \in Prod(\mathbf{T})$, $B \to^*_{\mathbf{T}} A$ (which implies $Term(\mathbf{T}) = \{A\}$).

The tile systems used in our constructions have $\tau = 2$ with the strength function ranging over $\{0, 1, 2\}$. It is known that $\tau = 1$ systems with strength function ranging over $\{0, 1\}$ are rather limited [7, 6].

[2] More formally,

$$\Gamma(t_1, t_2) = \begin{cases} g(bond_{D^{-1}}(t_1), bond_D(t_2)) \text{ if } \exists D \in \mathcal{D} \text{ s.t. } pos(t_1) = D(pos(t_2)); \\ 0 \text{ otherwise.} \end{cases}$$

3 Arbitrarily Scaled Shapes and Their Complexity

In this section, we introduce the model for the output of the self-assembly process used in this paper. Let S be a finite set of locations on $\mathbb{Z}\times\mathbb{Z}$. The adjacency graph $G(S)$ is the graph on S defined by the adjacency relation where two locations are considered adjacent if they are directly north/south, or east/west of one another. We say that S is a **coordinated shape** if $G(S)$ is connected.[3] The **coordinated shape of assembly** A is the set $S_A = \{(i,j) \text{ s.t. } A(i,j) \neq \textit{empty}\}$. Note that S_A is a coordinated shape because A contains a single connected component.

For any set of locations S, and any $c \in \mathbb{Z}^+$, we define a **c-scaling of S** as

$$S^c = \{(i,j) \text{ s.t. } (\lfloor i/c \rfloor, \lfloor j/c \rfloor) \in S\}.$$

Geometrically, this represents a "magnification" of S by a factor c. Note that a scaling of a coordinated shape is itself a coordinated shape: every node of $G(S)$ gets mapped to a c^2-node connected subgraph of $G(S^c)$ and the relative connectivity of the subgraphs is the same as the connectivity of the nodes of $G(S)$. A parallel argument shows that if S^c is a coordinated shape, then so is S. We say that coordinated shapes S_1 and S_2 are **scale-equivalent** if $S_1^c = S_2^d$ for some $c, d \in \mathbb{Z}^+$. Two coordinated shapes are **translation-equivalent** if they can be made identical by translation. We write $S_1 \cong S_2$ if S_1^c is translation-equivalent to S_2^d for some $c, d \in \mathbb{Z}^+$. Scale-equivalence, translation-equivalence and $\cong$ are equivalence relations. We call $\tilde{S}$, the equivalence class of coordinated shapes under "$\cong$", the **shape** $\tilde{S}$. We say that $\tilde{S}$ is the **shape of assembly** A if $S_A \in \tilde{S}$. The view of computation performed by the self-assembly process espoused here is the production of a shape as the "output" of the self-assembly process. Intuitively, the idea of scale-invariance attempts to formalize the notion that a physical object can be constructed from arbitrarily small pieces.

Having defined the notion of shapes, we turn to their descriptional complexity. As usual, the Kolmogorov complexity of a binary string x with respect to a universal Turing machine U is $K_U(x) = \min\{|p| \text{ s.t. } U(p) = x\}$. (See the exposition of Li and Vitanyi [9] for an in-depth discussion of Kolmogorov complexity.) Let us fix some "standard" universal machine U. We call the Kolmogorov complexity of a coordinated shape S to be the size of the smallest program outputting it as a list of locations:[4,5]

[3] We say "coordinated" to make explicit that a fixed coordinate system is used. We reserve the unqualified term "shape" for when we ignore scale and translation.

[4] Note that $K(S)$ is within an additive constant of $K(x)$ where x is some other effective description of S, such as a characteristic function. Since our results are asymptotic, they are independent of the specific representation choice. One might also consider invoking a two dimensional computing machine, but it is not fundamentally different for the same reason.

[5] Notation $\langle\cdot\rangle$ indicates some standard binary encoding of the object(s) in the brackets. In the case of coordinated shapes, it means an explicit binary encoding of the set of locations. Integers, tuples or other data structures are similarly given simple explicit encodings.

$$K(S) = \min\{|s| \text{ s.t. } U(s) = \langle S\rangle\}.$$

The Kolmogorov complexity of a shape $\tilde{S}$ is:

$$K(\tilde{S}) = \min\left\{|s| \text{ s.t. } U(s) = \langle S\rangle \text{ for some } S \in \tilde{S}\right\}.$$

We define the **tile-complexity** of a coordinated shape S and shape $\tilde{S}$ respectively as:

$$K_{sa}(S) = \min\left\{\begin{array}{l} n \text{ s.t. } \exists \text{ a tile system } \mathbf{T} \text{ of } n \text{ tile types that uniquely produces} \\ \text{assembly } A \text{ and } S \text{ is the coordinated shape of } A \end{array}\right\}$$

$$K_{sa}(\tilde{S}) = \min\left\{\begin{array}{l} n \text{ s.t. } \exists \text{ a tile system } \mathbf{T} \text{ of } n \text{ tile types that uniquely produces} \\ \text{assembly } A \text{ and } \tilde{S} \text{ is the shape of } A \end{array}\right\}.$$

4 Relating Tile-Complexity and Kolmogorov Complexity

The essential result of this paper is the description of the relationship between the Kolmogorov complexity of any scale-invariant shape and the number of tile types necessary to self-assemble it.

Theorem 1. *There exist constants a_0, b_0, a_1, b_1 such that for any shape $\tilde{S}$,*

$$a_0 K(\tilde{S}) + b_0 \le K_{sa}(\tilde{S}) \log K_{sa}(\tilde{S}) \le a_1 K(\tilde{S}) + b_1. \tag{1}$$

Note that since any tile system of n tile types can be described by $O(n \log n)$ bits, the theorem implies there is a way to construct a tiling system such that asymptotically at least a constant fraction of these bits is used to "describe" the shape rather than any other aspect of the tiling system.

Proof (of Theorem 1). To see that $a_0 K(\tilde{S}) + b_0 \le K_{sa}(\tilde{S}) \log K_{sa}(\tilde{S})$, realize that there exists a constant size program p_{sa} that, given a binary description of a tile system, simulates its self-assembly, making arbitrary choices where multiple tile additions are possible. If the self-assembly process terminates, p_{sa} outputs the coordinated shape of the terminal assembly as the binary encoding of the list of locations in it. Any tile system $\mathbf{T}$ of n tile types with any diagonal strength function and any threshold τ can be represented by a string $d_{\mathbf{T}}$ of $4n\lceil\log 4n\rceil + 16n$ bits: For each tile type, the first of which is assumed to be the seed, specify the bond types on its four sides. There are no more than $4n$ bond types. In addition, for each tile type $\boxed{t}$ specify for which of the 16 subsets $L \subseteq \mathcal{D}$, $\sum_{D \in L} g(bond_D(\boxed{t})) \ge \tau$. Note that the possibility of negative bond strengths is taken into account, but a diagonal strength function is assumed. If $\mathbf{T}$ is a tile system uniquely producing an assembly that has shape $\tilde{S}$, then $K(\tilde{S}) \le |p_{sa} d_{\mathbf{T}}|$. The left inequality in eq. 1 follows with the multiplicative constant $a_0 = 1/4 - \varepsilon$ for arbitrary $\varepsilon > 0$.

We prove the right inequality in eq. 1 by developing a construction (sketched in Section 5 and detailed in the full paper) showing how for any program s s.t.

$U(s) = \langle S \rangle$, we can build a tile system $\mathbf{T}$ of $64\lceil \frac{|p|}{\log |p|} \rceil + b$ tile types, where b is a constant and p is a string consisting of a fixed program p_{sb} and s (i.e. $|p| = |p_{sb}| + |s|$), that uniquely produces an assembly whose shape is $\tilde{S}$ such that $S \in \tilde{S}$. Program p_{sb} and constant b are both independent of S. The right inequality in eq. 1 follows with the multiplicative constant $a_1 = 64 + \varepsilon$ for arbitrary $\varepsilon > 0$. □

Our result can be used to show that the tile-complexity of shapes is uncomputable:

Corollary 1. *K_{sa} of shapes is uncomputable. In other words, the following language is undecidable: $\tilde{L} = \left\{ (l, n) \text{ s.t. } l = \langle S \rangle \text{ for some } S \in \tilde{S} \text{ and } K_{sa}(\tilde{S}) \leq n \right\}$.*

Language $\tilde{L}$ should be contrasted with $L = \{(l, n) \text{ s.t. } l = \langle S \rangle \text{ and } K_{sa}(S) \leq n\}$ which is decidable (but hard to compute in the sense of NP-completeness [1]).

Proof (of Corollary 1). We essentially parallel the proof that Kolmogorov complexity is uncomputable. If $\tilde{L}$ were decidable, then we can make a program that computes $K_{sa}(\tilde{S})$ and subsequently uses Theorem 1 to compute an effective lower bound for $K(\tilde{S})$. Then we can construct a program p that given n outputs some coordinated shape S (as a list of locations) such that $K(\tilde{S}) \geq n$ by enumerating shapes and testing with the lower bound, which we know must eventually exceed n. But this results in a contradiction since $p\langle n \rangle$ is a program outputting $S \in \tilde{S}$ and so $K(\tilde{S}) \leq |p| + \lceil \log n \rceil$. But for large enough n, $|p| + \lceil \log n \rceil < n$. □

5 Sketch of the Programmable Block Construction

5.1 Overview

The uniquely produced terminal assembly A of our tile system will consist of square "blocks" of c by c tiles. There will be one block for each location in S. Consider the coordinated shape in Fig. 1(a). An example assembly A is graphically represented in Fig. 1(b), where each square represents a block containing c^2 tiles. Self-assembly initiates in the *seed block*, which contains the seed tile, and proceeds according to the arrows illustrated between blocks. Thus if there is an arrow from one block to another, it indicates that the growth of the second block (a *growth block*) is initiated from the first. A terminated arrow indicates that the block does not initiate the self-assembly of an adjacent block in that direction. Fig. 1(c) describes our nomenclature: an arrow comes into a block on its input side, arrows exit on propagating output sides, and terminated arrows indicate terminating output sides. The seed block has four output sides, which can be either propagating or terminating. Each growth block has one input and three output sides, which are also either propagating or terminating.

The input/output connections of the blocks form a spanning tree rooted at the seed block. During the progress of the self-assembly of the seed block, a computational process determines the input/output relationships of the rest of

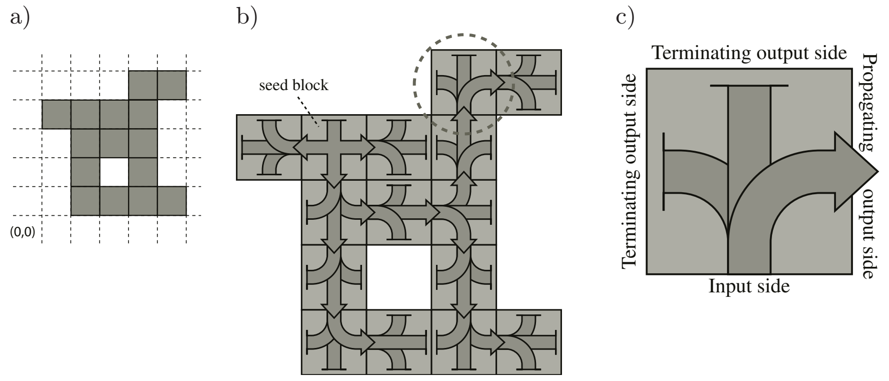


Fig. 1. Forming a shape out of blocks: a) A coordinated shape S. b) An assembly composed of c by c blocks that grow according to transmitted instructions such that the shape of the final assembly is $\tilde{S}$. Arrows indicate information flow and order of assembly. (Not drawn to scale.) The seed block and the circled growth block are schematically expanded in Fig. 2. c) The nomenclature describing the types of block sides

the blocks in the assembly. This information is propagated from block to block (along the arrows in Fig. 1(b)) during self-assembly and describes the shape of the assembly. By following the instructions each growth block receives in its input, the block decides where to start the growth of the next block and what information to pass to it in turn. The scaling factor c is set by the size of the seed block. The computation in the seed block ensures that c is large enough that there is enough space to do the necessary computation within the other blocks.

We present a sketch of a general construction that represents a Turing–universal way of guiding large scale self-assembly of blocks based on an input program p. In the following section, we describe the architecture of seed and growth blocks on which arbitrary programs can be executed. In section 5.3 we discuss the programming of p that is required to grow the blocks in the form of a specific shape. For a complete presentation of our construction, including the proof that the terminal assembly is uniquely produced, see the full version of this paper.

5.2 Architecture of the Blocks

The internal structure of a growth block is shown in Fig. 2(a). The first part is a Turing machine simulation, which is based on [12, 7]. The machine simulated is a universal Turing machine that takes its input from the propagating output side of the previous block. This TM has an output alphabet $\{0, 1, S\}^3$ and an input alphabet $\{(000), (111)\}$ on a two-way tape. The output of the simulation, as 3-tuples, is propagated until the diagonal. The diagonal propagates each member of the 3-tuples crossing it to one of the three output sides, like a prism separating the colors of the spectrum. This allows the single TM simulation to produce

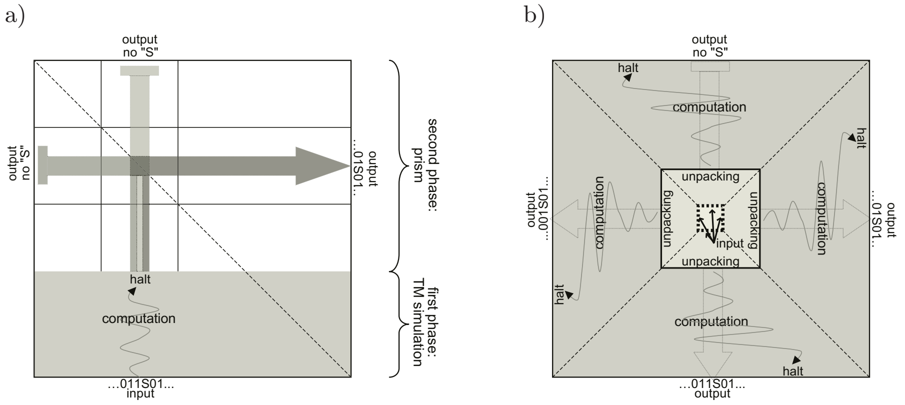


Fig. 2. Internal structure of a growth block (a) and seed block (b)

three separate strings targeted for the three output sides. The "S" symbol in the output of the TM simulation is propagated like the other symbols. However, it acts in a special way when it crosses the boundary tiles at the three output sides of the block, where it starts a new block. The output sides that receive the "S" symbol become propagating output sides and the output sides that do not receive it become terminating output sides. Obviously, the TM simulation decides which among the three output sides will become propagating output sides, and what information they should contain, by outputting appropriate tuples. Subsequent blocks will use this information as a program, as discussed in section 5.3.

An outline of the seed block is shown in Fig. 2(b). While growth blocks contain a single TM simulation that outputs a different string to each of the three output sides, the seed block contains four *identical* TM simulations that output different strings to each of the four output sides. This is possible because the border tile types transmit information selectively: the computation in the seed block is performed using 4-tuples as the alphabet in a manner similar to the growth blocks, but only one of the elements of the 4-tuple traverses the border of the seed block. As with growth blocks, if the transmitted symbol is "S", the outside edge initiates the assembly of the adjoining block. The point of having four identical TM simulations is to ensures that the seed block is square: while a growth block uses the length of its input side to set the length of its output sides (via the diagonal), the seed block does not have any input sides. (Remember that it is the seed block that sets the size of all the blocks.)

The initiation of the Turing machine simulations in the seed block is done by tile types encoding the program p that guides the block construction. The natural approach to provide this input is using 4 rows (one for each TM) of unique tiles encoding one bit per tile; however, this does not result in an asymptotically optimal encoding. We follow Adleman et al [3] and encode on the order of $\log n/\log\log n$ bits per tile where n is the length of the input. This representation is then unpacked into a one-bit-per-tile representation used by the

TM simulation (Fig. 2(b)). Adleman et al's method requires $O(n/\log n)$ tiles to encode n bits, leading to the asymptotically optimal result of Theorem 1.

5.3 Programming Blocks and the Value of the Scaling Factor c

In order for our tile system to produce some assembly whose shape is $\tilde{S}$, instructions encoded in p must guide the construction of the blocks by deciding on which side of which block a new block begins to grow and what is encoded on the edge of each block. For our purposes, we take $p = p_{sb}\langle s\rangle$ (i.e. p_{sb} takes s as input), where s is a program that outputs the list of locations in the shape S. p_{sb} runs s to obtain this list and plans out a spanning tree t over these locations (it can just do a depth-first search) starting from some arbitrarily chosen location that will correspond to the seed block. The information passed along the arrows in Fig. 1(b) is $p_{gb}\langle t,(i,j)\rangle$ which is the concatenation of a program p_{gb} to be executed within each growth block, and an encoding of the tree t and the location (i,j) of the block into which the arrow is heading. When executed, $p_{gb}\langle t,(i,j)\rangle$ evaluates to a 3-tuple encoding of $p_{gb}\langle t, D(i,j)\rangle$ together with symbol "S" for each propagating output side D. Thus, each growth block passes $p_{gb}\langle t, D(i,j)\rangle$ to its Dth propagating output side as directed by t. Note that program p_{sb} in the seed tile must also run long enough to ensure that c is large enough that the computation in the growth blocks has enough space to finish without running into the sides of the block or into the diagonal. Nevertheless, the scaling factor c is dominated by the building of t in the seed block, as the computation in the growth blocks takes only $poly(|S|)$. Since the building of t is dominated by the running time of s, we have $c = poly(time(s))$.

6 Discussion and Open Questions

Because the Kolmogorov complexity of a string depends on the universal Turing machine chosen, the complexity community adopted a notion of additive equivalence, where additive constants are ignored. However, Theorem 1 includes multiplicative constants as well, which are not customarily discounted. It might be possible to use a more clever method of unpacking (Sect. 5.2) and a seed block construction that reduces the multiplicative constant a_1 of Theorem 1. Correspondingly, there might be a more efficient way to encode any tile system than described in the proof of the theorem, and thereby increase a_0.

It is most likely that our block construction can be easily adapted to use a different encoding of the input, leading to a different form of Theorem 1 for variations on the Tile Assembly Model. Recent work by Aggarwal et al [4] shows that allowing non-diagonal strength functions allows a dramatic change in tile complexity.

The programmable block construction may have other applications as well. For instance, it is easy to reprogram it to simulate, using few tile types, a large deterministic $\tau = 1$ tile system for which a short algorithmic description of the tile set exists. We believe a slightly extended version of the block construction can also be used to provide compact tile sets that simulate other $\tau = 2$ tile systems that have short algorithmic descriptions.

The scaling factor $c = poly(time(s))$ is extremely large since $|S|$ is presumably enormous for cases where our results are of interest and s must output every location in S. Are there special instances where it is not necessary to run the program s outputting S to completion but query it in a few locations relevant to the immediate block being built?

Acknowledgements. We thank Len Adleman, members of his group, and Paul Rothemund for fruitful discussions and suggestions. We thank Rebecca Schulman and David Zhang for useful and entertaining conversations about descriptional complexity of tile systems. This work was supported by NSF CAREER Grant No. 0093486.

References

1. L. Adleman, Q. Cheng, A. Goel, M.-D. Huang, D. Kempe, P. M. de Espanes, and P. W. K. Rothemund. Combinatorial optimization problems in self-assembly. In *Proc. of STOC*, 2002.
2. L. M. Adleman. Toward a mathematical theory of self-assembly (extended abstract). Technical report, University of Southern California, 1999.
3. L. M. Adleman, Q. Cheng, A. Goel, and M.-D. A. Huang. Running time and program size for self-assembled squares. In *ACM Symposium on Theory of Computing*, pages 740–748, 2001.
4. G. Aggarwal, M. Goldwasser, M. Kao, and R. T. Schweller. Complexities for generalized models of self-assembly. In *Symposium on Discrete Algorithms*, 2004.
5. M. Cook, P. W. K. Rothemund, and E. Winfree. Self-assembled circuit patterns. In *DNA Based Computers 9*, pages 91–107, 2004.
6. P. W. K. Rothemund. *Theory and Experiments in Algorithmic Self-Assembly.* PhD thesis, University of Southern California, Los Angeles, 2001.
7. P. W. K. Rothemund and E. Winfree. The program-size complexity of self-assembled squares (extended abstract). In *ACM Symposium on Theory of Computing*, pages 459–468, 2000.
8. T. H. LaBean, H. Yan, J. Kopatsch, F. Liu, E. Winfree, J. H. Reif, and N. C. Seeman. Construction, analysis, ligation, and self-assembly of DNA triple crossover complexes. *Journal of the Americal Chemical Society*, 122:1848–1860, 2000.
9. M. Li and P. Vitanyi. *An Introduction to Kolmogorov Complexity and Its Applications.* Springer, second edition, 1997.
10. C. Mao, T. H. LaBean, J. H. Reif, and N. C. Seeman. Logical computation using algorithmic self-assembly of DNA triple-crossover molecules. *Nature*, 407:493–496, 2000.
11. C. Mao, W. Sun, and N. C. Seeman. Designed two-dimensional DNA holliday junction arrays visualized by atomic force microscopy. *Journal of the Americal Chemical Society*, 121:5437–5443, 1999.
12. R. M. Robinson. Undecidability and nonperiodicity of tilings of the plane. *Inventiones Mathematicae*, 12:177–209, 1971.
13. H. Wang. Proving theorems by pattern recognition. II. *Bell Systems Technical Journal*, 40:1–42, 1961.
14. E. Winfree. Simulations of computing by self-assembly, Caltech CS TR 1998.22.

15. E. Winfree. *Algorithmic Self-Assembly of DNA*. PhD thesis, California Institute of Technology, Pasadena, 1998.
16. E. Winfree, F. Liu, L. A. Wenzler, and N. C. Seeman. Design and self-assembly of two dimensional DNA crystals. *Nature*, 394:539–544, 1998.

Aqueous Computing with DNA Hairpin-Based RAM

Naoto Takahashi[1], Atsushi Kameda[2], Masahito Yamamoto[3], and Azuma Ohuchi[3]

[1] Graduate School of Engineering, Hokkaido University, North 13, West 8, Kita-ku, Sapporo, Hokkaido 060-8628, Japan
[2] CREST, Japan Science and Technology Agency (JST), Honmachi 4-1-8, Kawaguchi, Saitama 332-0012, Japan
[3] CREST, Japan Science and Technology Agency (JST) and Graduate School of Information Science and Technology, Hokkaido University, North 14, West 9, Kita-ku, Sapporo, Hokkaido 060-0814, Japan
{naoto, kameda, masahito, ohuchi}@dna-comp.org
http://ses3.complex.eng.hokudai.ac.jp/

Abstract. DNA RAM can eliminate the computational overhead of sequence design because the same RAM can be used for various computations once it is made. In this paper, we report a new method to construct RAM by using the hairpin structure of DNA, and the 4-bit RAM that we actually constructed with this method. We conducted an experiment to distinguish all 16 states of the 4-bit RAM, and another to verify our achievement of a successive writing operation. Finally, we performed aqueous computing with this 4-bit RAM.

1 Introduction

In the conventional computational methods of DNA computing, researchers had to design and make up the complete set of DNA sequences to obtain the solutions of each given problem. Because designing the sequence can be more difficult than solving the given problem, this can be a serious bottleneck. However, this overhead may be eliminated if we can use the same sequences to solve the various problems. This is achieved with Random Access Memory (RAM). Once RAM has been made with DNA, it can be widely used for various computations.

Read Only Memory (ROM) can be considered as another kind of memory. An associative memory, reported by Baum [1] and Reif et al. [2], is one kind of ROM. It can be used to perform the associative searches, and it exploits the capability of DNA effectively. Nested Primer Molecular Memory was developed by Kashiwamura et al. [3], and it is an addressing memory that uses Polymerase Chain Reaction (PCR). If we apply the primers in the correct order when PCR, data becomes readable. This is another kind of ROM. Chen et al. studied a DNA memory with learning and associative recall [4]. This memory changes its own sequence by digestion and extension, thus it can learn a segment of unknown input DNA. Using that learned memory, researchers can search for similar strands to the initial input DNA in other unknown DNA. Much research has been conducted on DNA memory, and we focused on RAM because of DNA's versatility.

To create DNA RAM, bits have to be represented by using DNA, and three earlier methods have been investigated. The first was the Cut, Elongate, and Ligate (CEL) method developed by Head et al. [5]. This method uses several restriction enzymes, polymerase, ligase, and circular double-strand DNA. The second was the Cut, Delete, and Ligate (CDL) method also developed by Head et al. [6]. This method also uses several restriction enzymes, ligase, and circular double-strand DNA, but needs pairs of the same kind of restriction sites to represent a bit. The third was the method of using Peptide Nucleic Acid (PNA) developed by Yamamura et al. [7]. This method uses single-strand PNA and DNA.

We propose using hairpin structure DNA to represent bits. This hairpin method does not use enzymes, and does not require regulating solution temperature. The only materials required for this method is DNA. Therefore, it may be an easier method to handle than the earlier methods described above. We explain hairpin structure in the next section.

Aqueous computing is an application that can make good use of the nature of DNA RAM, and its framework was suggested by Head [8]. Because DNA RAM exists in a solution randomly, it is easy to split into several processes with equal component percentage, and to mix them uniformly is easy, also. Using this aqueous nature with ingenuity can create the solution candidates. Further details about an algorithm for this is illustrated with one concrete example in section 5.

In this paper, we propose a method for creating RAM by exploiting the hairpin structure of DNA. Next, we show our experimental results of the constructing this RAM, distinguishing its states, and using it for successive writing operation.

2 DNA Hairpin-Based RAM (DNA-HRAM)

2.1 DNA Hairpin

The structure of a single-strand DNA that has bonded to itself is like that of a hairpin. This is shown in Fig. 1 in what we call the "closed" state. Interestingly, the bonded stem can be separated by one single-strand DNA that is complementary to the lead and stem. This single-strand DNA is called an opener. As Fig. 2 shows, an opener bonds to the lead first and then bonds to the stem, and we call this state "open". The closed state is counted as 1, and the open state is counted as 0. In this way, we can represent a bit, and this RAM is called DNA Hairpin-based RAM (DNA-HRAM). The only experimental treatment that we have to perform to open a hairpin is to mix in an opener. Regulating the solution temperature is not necessary because this opening reaction occurs at room temperature. Moreover, basically, the only materials required for this method are DNA strands. Additionally, DNA-HRAM is reusable because the opener can be separated from the hairpin by raising the solution temperature.

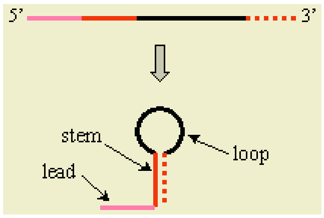


Fig. 1. "Closed" state

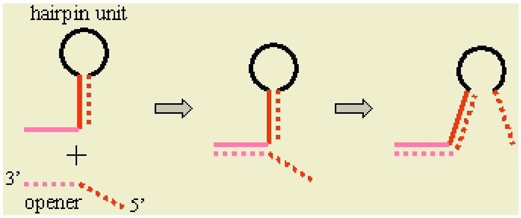


Fig. 2. "Open" state and opener

To be precise, a hairpin consists of a stem, loop, and complementary stem, but we add a lead to the 5'-end of a hairpin to obtain an opening reaction. The hairpin and lead are collectively called a hairpin unit.

2.2 Multi-bit DNA-HRAM

Multi-bit DNA-HRAM is achieved by splicing the multiple hairpin units previously described. We constructed 4-bit DNA-HRAM as shown in Fig. 3. The sequences of this 4-bit DNA-HRAM and its openers are shown in Table 1. The lead is 20 mer, the stem is 20 mer, the loop is 7 mer, and the complementary stem is 20 mer. Accordingly, the total length of one hairpin unit is 67 mer, that of one opener is 40 mer, and that of one target sequence is 288 mer.

Table 1. Sequences (in order of from 5' to 3')

Name	Sequence
lead A	TCCAGATTAAATGGAGTGGG
hairpin A	GGTTGGGAGCGATATTCGTA TTCATCC TACGAATATCGCTCCCAACC
lead B	TAGCAAAAGCGGCCTCACTC
hairpin B	CCAGTATATACCAGCACCTG TTAGCCC CAGGTGCTGGTATATACTGG
lead C	GCAGCAAGAATAGTTCACCT
hairpin C	CAGAACGACATTACGAGAGA CGCTGTC TCTCTCGTAATGTCGTTCTG
lead D	ATGTGGTTAAGCAACTGCCG
hairpin D	CTGGAATACTCTTAGGCTGG GTTCAGT CCAGCCTAAGAGTATTCCAG
lead E	GCATGCCAGACCTAAGGATG
opener A	TACGAATATCGCTCCCAACC CCCACTCCATTTAATCTGGA
opener B	CAGGTGCTGGTATATACTGG GAGTGAGGCCGCTTTTGCTA
opener C	TCTCTCGTAATGTCGTTCTG AGGTGAACTATTCTTGCTGC
opener D	CCAGCCTAAGAGTATTCCAG CGGCAGTTGCTTAACCACAT

In the sequence design step, each hairpin can only be opened by its own opener, and taking note of that is important. That is to say, the only opener that can open hairpin A is opener A. We referred to the recent research by Uejima and Hagiya [9] to resolve this sequence design problem.

This hairpin mechanism was also used for conformational addressing by Kameda et al. [10]. They found that the memory becomes readable if openers are mixed in the correct sequence.

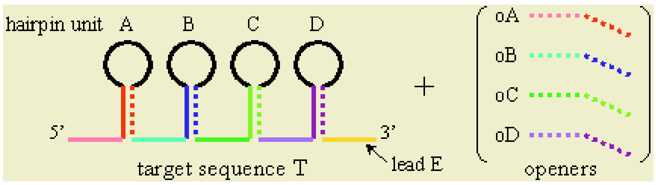


Fig. 3. Target sequence and openers

3 Construction

To construct this target sequence, we had to decompose it into several short parts and then recompose them because directly synthesizing a long DNA sequence is very difficult. The composition of these parts was reported in our previous study [11].

To examine whether the target sequence was actually composed, we used Poly-Acrylamide Gel Electrophoresis (PAGE). The sequence in which part A is spliced to part B is named A+B. The migration distance of A+B is different from that of A and that of B because A+B has a different length than A and B. By the same token, if a new band that is not made up of its parts or intermediate products appears when all parts are mixed, it can be considered as a product in which all parts are combined. Accordingly, A+B+C+D+E should have a different band from all the others, and it should be the target sequence that is shown in Fig. 3. The results of this experiment are in Fig. 4, and the completion of the target sequence is indicated by the arrow. In this experiment, DNA was stained with SYBR Gold after PAGE.

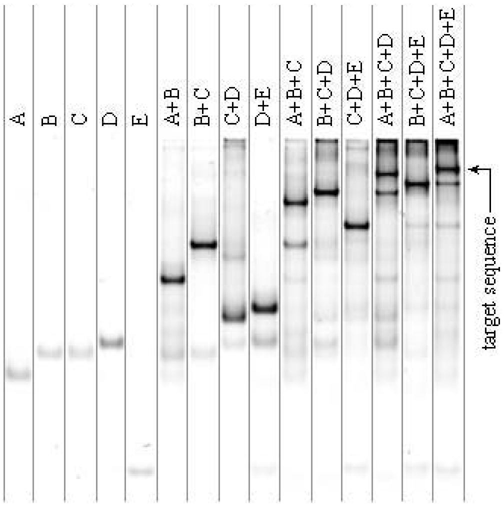


Fig. 4. Experimental results of constructing 4-bit DNA-HRAM

4 Distinguishing States

We checked whether each bit of this 4-bit DNA-HRAM could be opened independently and specifically, and the results are in Fig. 5. Here, we used PAGE again. The difference of migration distance is caused not only by the length of the sequence but also by its form. That is, sequences that have different forms are carried different distances even if they have same length. For example, the state where no openers are mixed, that is, T in Fig. 5, was carried a different distance from the state where opener A was mixed, that is T+oA, because each had different forms. By the same token, all 16 states were carried to different positions. The states T+oB, T+oC, and T+oD definitely have different band positions. These results mean that all 4 hairpins were opened independently and specifically by their individual openers, even though, some band positions of all 16 states were very close. This may be unavoidable because of the resolutive limits of this method. However, the true need is detecting one solution after computing and not distinguishing all states. Thus, we propose one idea in section 6. Unfortunately, some lanes had more than one band. That means the reaction for opening the hairpin was not perfect.

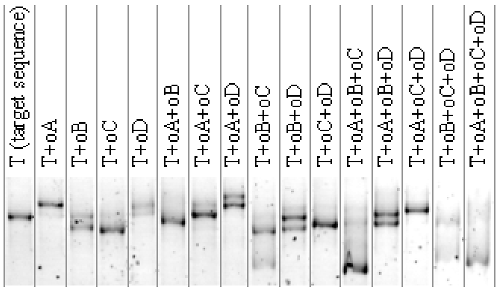


Fig. 5. Experimental results of distinguishment of all 16 states

5 Retaining Bit Pattern

To accomplish the successive writing operation, such as those shown in Fig. 8, retaining the bit pattern of RAM after mixing the split solutions is necessary. Basically, openers must be mixed in large enough quantities so that the opening reaction occurs effectively. Accordingly, a lot of openers that have not reacted remain in the solution. Because some DNA-HRAM that are from another solution and that have different states may be opened by those openers remaining after mixing, all remaining openers have to be eliminated from the solution before mixing. Next, we explain how to eliminate them, and show the experimental results of using that elimination treatment for a successive writing operation.

5.1 Eliminating Opener

We adopted use of a cover and magnetic beads to eliminate the remaining openers. The elimination technique is shown in Fig. 6. The cover is the same sequence as the lead, but its 5'-end is biotinylated. After the reaction of opening hairpins, we mix covers in sufficient quantities, and they hybridize with the remaining openers. Next, we mix the appropriate amount of binding buffer (20 mM Tris-Cl, 1.0 M NaCl, 1 mM EDTA, 0.02% Triton X-100; pH 7.8) and streptavidin with the solution. The covers that bonded to openers are bound to streptavidin because of the 5'-end biotin. The remaining covers are also bound to streptavidin. Since streptavidins are bound to magnetic beads, they can be separeted with a magnet. Thus, we can achieve the solution that contains no magnetic beads, resultingly the remaining openers and covers are eliminated with them.

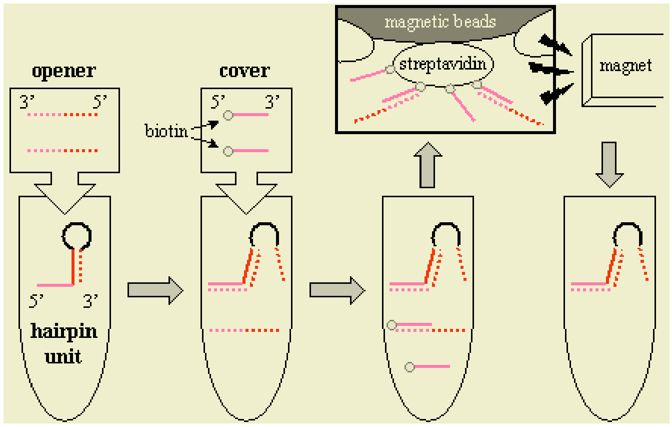


Fig. 6. Elimination process

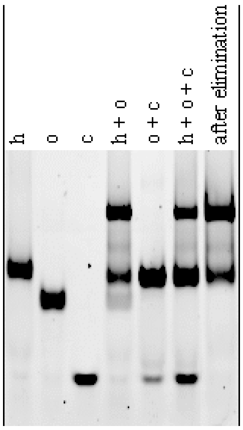


Fig. 7. Experimental results of elimination

The results of this test experiment is shown in Fig. 7. In each lane, the index h means hairpin unit, the index o means opener, and the index c means cover. When the hairpin unit and opener were mixed, a new band appeared, the lane of h+o, and that is the band of opened hairpin units. Since the amount of openers were larger than that of hairpin units, the band of openers remained. However, some closed hairpin units also remained in the lane because the reaction of hairpin opening, as mentioned earlier, can not be performed perfectly. When opener and cover were mixed, that is the lane o+c, most openers bonded to covers, thus a new band appeared. That the band positions of h and o+c happen to be very close should be noted. In the lane of h+o+c, cover was added after mixing the hairpin unit and opener. Three bands appear to be in this lane, but the middle band includes two kinds of strands, h and o+c. The other high and low bands are that of h+o and c. We applied the elimination described above to this solution, and the far right lane is the result of that. The thickness of the middle band decreased. This means that o+c was eliminated and h remained. Additionally, band c disappeared, and there were no remaining bands except h+o and h. Therefore, we verified that the remaining openers and covers were actually eliminated.

5.2 Successive Operation

We performed the aqueous computing that is shown in Fig. 8 for the verification of the successive writing operation with the 4-bit DNA-HRAM.

This is an instance of Maximum Independent Set Problem (MISP), and it is such as shown in Fig. 9. If no two vertices in a subset of a graph G are connected by an edge, the subset is an independent set of G. Therefore, an independent set containing the largest possible number of vertices is a maximum independent set. MISP is the problem of obtaining a maximum independent set, and it belongs to the class of NP-complete.

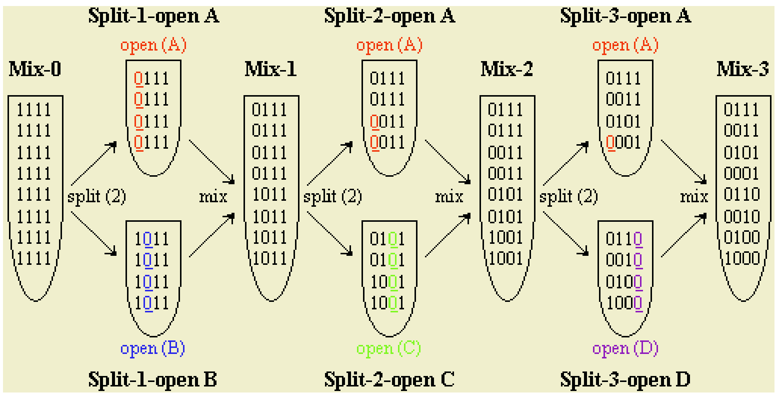


Fig. 8. Algorithm and process of aqueous computing

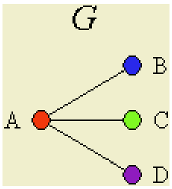


Fig. 9. Graph of solved Maximum Independent Set Problem

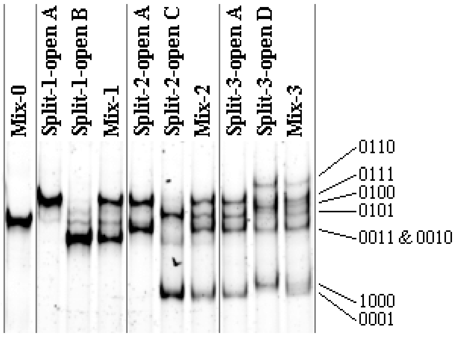


Fig. 10. Experimantal results of successive writing operation

Here, we explain how to solve a MISP with an aqueous algorithm. The idea is very simple. First, we prepare a lot of subsets containing all the vertices. Then, we split them into two processes. In one process, to resolve connected vertices, one vertex that is either side of an edge is removed from the subsets. In another process, the vertex of another side is removed. After mixing these two processes, the connection of an edge has been cleared away. We repeat these splits and mixes until all the connections are cleared. Thus, this algorithm makes solving the problem possible in linear chemical steps even if it is a large scale combinatorial optimization problem. Besides, the only required operation is to remove the vertex.

Experimental results of the successive writing operation are shown in Fig. 10. The bit state 1 indicates the closed state of the hairpin, and 0 indicates the open state. Starting from the left, each bit corresponds to hairpin A, B, C, and D. For example, 0101 means that hairpin A is open, B is closed, C is open, and D is closed. The split solutions were sampled after elimination of openers.

Mix-0 is the initial state in which all the hairpins of 4-bit DNA-HRAM are closed, so only one band of 1111 appears in this lane.

First, Mix-0 was split into two solutions, to resolve the edge between vertices A and B. In one solution, hairpin A was opened, and the bit pattern was changed from 1111 to 0111. This was the lane of Split-1-open A, and only one band of 0111 was obtained. In another solution, hairpin B was opened, and the bit pattern was changed from 1111 to 1011. This was the lane of Split-1-open B, and only one band of 1011 was obtained. After eliminating openers, these two solutions were mixed as Mix-1. Most DNA-HRAM were clearly divided into two states, though a few 1111 remained. That means the elimination of openers was successful.

Next, Mix-1 was split into two solutions, to resolve the edge between vertices A and C. In one solution, hairpin A was opened, and the bit pattern was changed from 1011 to 0011. Because hairpin A of 0111 had been already opened, it retained its bit pattern. This was the lane of Split-2-open A, and two bands of 0111 and 0011 were obtained. In another solution, hairpin C was opened, and the bit patterns were changed from 0111 and 1011 to 0101 and 1001, respectively. This is the lane of Split-2-open C, and two bands of 0101 and 1001 were obtained. After eliminating openers, these two solutions were mixed as Mix-2. Here, these DNA-HRAM were very clearly divided into four states.

Then, Mix-2 was split into two solutions, to resolve the edge between vertices A and D. In one solution, hairpin A was opened, and the bit pattern was changed from 1001 to 0001. Because hairpin A of 0111, 0011, and 0101 had been already opened, they retained their bit patterns. This is the lane of Split-3-open A, and four bands of 0111, 0011, 0101, and 0001 were obtained. However, as shown in Fig. 10, the lane of Split-3-open A seems similar to that of Mix-2 because the band positions of 1001 and 0001 are very close. In another solution, hairpin D was opened, and the bit patterns were changed from 0111, 0011, 0101, and 1001 to 0110, 0010, 0100, and 1000. This is the lane of Split-3-open D, and the four bands of 0110, 0010, 0100, and 1000 were obtained. After eliminating openers, these two solutions were mixed as Mix-3. Actually, we wanted eight bands here, but we could observe only seven bands. The reason is that the band positions of 0011 and 0010 are very close. Therefore, we believe that these DNA-HRAMs contain eight different bit patterns, and that their computational power has been demonstrated.

6 Detecting Solution

One remaining problem is how to detect the solution. We created the solution candidates, and in the next step, we had to extract the optimum solution from among those. In the successive writing operation experiment, the bit patterns of each appeared band were determined by comparing them with the results of experimentally distinguishing states. However, that way may be impractical when the number of all bits increased. Here, we introduce an idea that of using the complementary strands of DNA-HRAM to extract the solution. An outline of this idea is shown in Fig. 11.

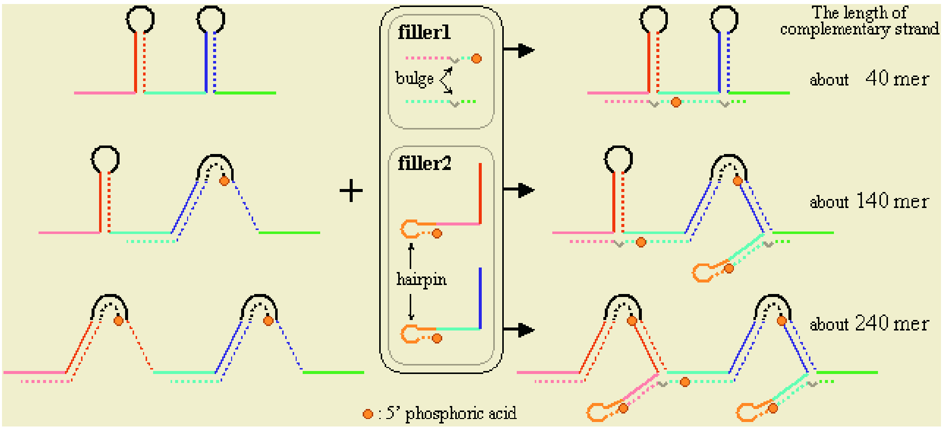


Fig. 11. Idea of Extracting Solution

This is an example of a 2-bit DNA-HRAM. Here, we want to obtain a solution from the three states shown on the left side. First, we mixed filler 1 with DNA-HRAM and performed ligation. Second, we added filler 2 and repeated the ligation process. The fillers bonded to a single-strand of each DNA-HRAM and were spliced as shown on the right side. As a result, we made three complementary strands of different lengths. In the case of MISP, the shortest complementary strand represents the solution. Finally, by using denatured PAGE, we were able to separate and identify the solution. However, in this method the opener strands must be changed to achieve effective ligation, and the length of each part must be coordinated. Currently, we have succeeded in the ligation of filler 1, as shown in the upper right. To read the bit pattern, we can use fluorescent materials.

7 Conclusion

We developed DNA-HRAM, and actually constructed 4-bit DNA-HRAM. We then checked whether each bit of this 4-bit DNA-HRAM could be opened independently and specifically, and found that all these bits had worked normally. Additionally, we achieved successive writing operations, and performed aqueous computing with this 4-bit DNA-HRAM. Finally, we also offered a possible way of extracting a solution.

We would like to make DNA-HRAM more efficient since there are still many things that we can improve. But, we must first bring the detection method to completion. In future work, we plan to try constructing a large-bit DNA-HRAM. We consider that the construction will not be so difficult now that we have the sequence set for 20-bit DNA-HRAM.

References

1. E. B. Baum, "Building an Associative Memory Vastly Larger Than the Brain", Science, Vol. 268, pp. 583-585, 1995.
2. J. H. Reif, T. H. LaBean, M. Pirrung, V. S. Rana, B. Guo, C. Kingsford, G. S. Wickham, "Experimental Construction of Very Large Scale DNA Databases with Associative Search Capability", 7th International Workshop on DNA-Based Computers (DNA7), in Lecture Notes in Computer Science 2340, pp. 231-247, 2002.
3. S. Kashiwamura, M. Yamamoto, A. Kameda, T. Shiba, A. Ohuchi, "Hierarchical DNA Memory Based on Nested PCR", 8th International Workshop on DNA-Based Computers (DNA8), in Lecture Notes in Computer Science 2568, pp. 112-123, 2002.
4. J. Chen, R. Deaton, Y. Wang, "A DNA-based Memory with In Vitro Learning and Associative Recall", Preliminary Proceedings of Ninth International Meeting on DNA Based Computers (DNA9), pp. 127-136, 2003.
5. T. Head, M. Yamamura, S. Gal, "Aqueous Computing: Writing on Molecules", Proceedings of the Congress on Evolutionary Computation, pp. 1006-1010, 1999.
6. T. Head, G. Rozenberg, R. S. Bladergroen, C. K. D. Breek, P. H. M. Lommerse, H. P. Spaink, "Computing with DNA by operating on plasmids", BioSystems, Vol. 57, pp. 87-93, 2000.
7. M. Yamamura, Y. Hiroto, T. Matoba, "Another Realization of Aqueous Computing with Peptide Nucleic Acid", 7th International Workshop on DNA-Based Computers (DNA7), in Lecture Notes in Computer Science 2340, pp. 213-222, 2002.
8. T. Head, "Circular Suggestions for DNA Computing", submitted to World Scientific, 1999.
9. H. Uejima, M. Hagiya, "Secondary Structure Design of Multi-state DNA Machine Based on Sequential Structure Transitions", Preliminary Proceedings of Ninth International Meeting on DNA Based Computers (DNA9), pp. 80-91, 2003.
10. A. Kameda, M. Yamamoto, H. Uejima, M. Hagiya, K. Sakamoto, A. Ohuchi, "Conformational addressing using the hairpin structure of single-strand DNA", Preliminary Proceedings of Ninth International Meeting on DNA Based Computers (DNA9), pp. 197-201, 2003.
11. N. Takahashi, A. Kameda, M. Yamamoto, A. Ohuchi, "Construction and verification of DNA hairpin-based RAM", Proceedings of The Ninth International Symposium on Artificial Life and Robotics (AROB 9th '04), Vol. 2, pp. 388-391, 2004.

A Programmable Molecular Computer in Microreactors

Danny van Noort

Ecology and Evolutionary Biology,
Princeton University, Princeton, NJ 08544, USA
Present address: Biointelligence Lab.,
School. of Computer Science and Engineering,
Seoul National University, San 56-1, Sinlim-dong,
Gwanak-gu, Seoul 151-742, Korea
danny@bi.snu.ac.kr

Abstract. To solve problems with DNA in microreactors, static systems are not favourable due to programmability. It will be shown that solving Boolean logic with a programmable microfluidic system is a step forward in molecular computing. Not only the use of DNA as information carrier is important but also the integration of electronic computing to control the system and readouts. This paper will show that the optimal configuration of computing in microfluidics is a system containing components analogue with electronic elementary building blocks components, such as switches, logic gates and memory, making it programmable for any type of Boolean problem.

1 Introduction

It has been foreseen that conventional electronic processors will reach its speed and size limit in the near future. This is the reason why intensive research has been started on alternative computing methods like fluidic computing [1], quantum computing [2] and DNA-computing [3]. The latter is the most promising because it is closely connected to the biological world and therefore will have applications in biotechnology, such as medical diagnostics and drug lead-compound optimisation. The research on operations with biomolecules can give a better insight into biological systems, while the information processing and construction abilities at molecular level can give new computing paradigms. However, digital computers have their advantages as well, especially as controller. Therefore an optimal computing solution would be to integrate electronic with molecular computing.

As shall be shown, microfluidic networks can be incorporated as an information carrier in a computing scheme. The advantages of microfluidics are the small volumes (in the pico-liter range) of DNA solution needed and the speed of reactions. With fluidic switches, such as micro valves, and micro pumps the flow can be (re-) directed. By controlling the sequence of switch settings with an electronic controller, a specific programme can be executed. The channels are like the wires in electronic circuits, transporting the information from one operator to another, to fluidic flip-flops, *i.e.* logical operators. Logical gates with biological components, so far, have been introduced in cells [4], biomolecules [5] and bioelectronics, such as nanowires [6].

Gehani and Reif [6], and later McCaskill and Wagler [7], came with a concept for re-configurable microreactors as a way to dynamically change the flows in microfluidic networks. However, these visions were still too futuristic and have no short term application. In this paper we introduce microfluidic logic operators, simple fluidic switches and memory. Furthermore, the use of electronic fluidic control components in microfluidic systems will be demonstrated in such way as to perform dynamic operations and programming. Finally a proposal for an actual fluidic computer will be made which in first instance solves a series of logical operations, for example, the Maximum Clique [9] and is more efficient than previous developed DNA-computers because of its flexible programming capability.

2 Selection Procedure

Selections can be made by using capture probes (CP), such as short selection DNA, to select longer single DNA strands containing coded information in the sequence of its base-pairs from a sequence space $\{S_i\}$. Hybridisation between these two is a selection, a YES or NO, *i.e.* a logic operation.

There are two ways of performing selections, using varied techniques: positive or negative selection. Positive selection retains S_k from the sequence space $\{S_i\}$ while negative selection discards S_k. One method of positive selection was proposed by McCaskill [9], introducing active transport of DNA by means of paramagnetic beads. The CPs are immobilised to the beads and the selected DNA is moved from one flow to another. Negative selection is reached by immobilising the CPs to a surface (such as beads) which hybridise to S_k, while the rest of the sequences ($S_{1,\ldots,k-1,\ k+1,\ldots,N}$) continue to next operator [10].

The advantage of positive selection is that there is a minimal amount of unwanted DNA, while the advantage of negative selection is the simplicity of operation and control, however with higher error rate. In the latter case it is possible to re-run the solution over the (regenerated) CPs as to optimise the purity of the DNA template solution.

3 Principle of Operation

It is clear from the above that logic operators can be defined with these selection procedures. A NOT operation corresponds to a negative selection, while the retention of a certain member (S_k) from the sequence space corresponds to a positive selection.

For now, we leave the selection procedure a black box with the operation $\boldsymbol{a}$ being a positive selection and $\neg\boldsymbol{a}$ a negative selection procedure (Fig 1a), since it is not important for the concept to know how the selection is precisely done. Two selectors in sequence will perform an AND operation, while two selectors in parallel will perform an OR operation (see Fig 1b).

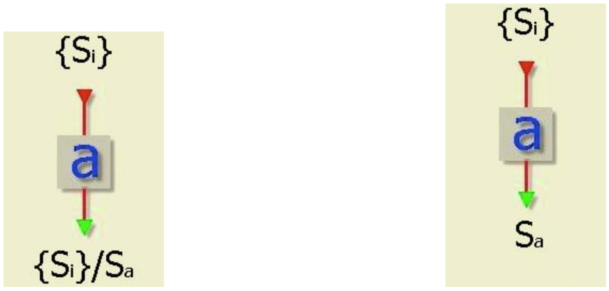
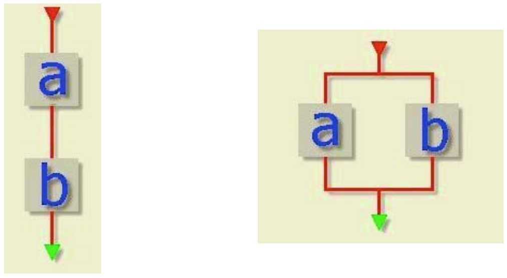


Fig. 1a. NOT operations on member **a** in a sequence space $\{S_i\}$ is made by a negative selection, while selection of member **a** (S_a) is made by a positive selection



a ∧ b a v b

Fig. 1b. AND operations can be made by having two selectors in sequence, while an OR operation is made by two selectors in parallel

By making a combination of a number of selectors in a certain connection, it is possible to solve Boolean logic problems.

4 Memory

Using switchable microfluidics makes it is possible to construct memories. Information can be stored in separate channels which hold the solutions with DNA, or other molecules used as information carriers. Figure 2 shows the schematics of the memory. A number of channels, determining the memory size, are connected to an input and output channel by multi-selection valves. The position of the valve gives

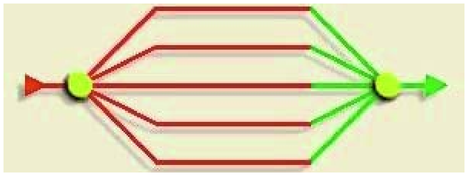


Fig. 2. The schematics of a memory with 5 channels and selection valves (the light grey circles)

the memory location (M_i, with i the number of channels). This would be similar to Direct Accessible Memory with read and write capabilities. The advantage of a memory is that information can be stored and retrieved at a later period. Iterative processes would then be possible, broadening the range of problem solving.

5 The Configurable Microfluidic Computer

To make a problem solving fluidic network feasible, a series of switches must be incorporated into the design of the microsystem. This allows us to change the direction of the flows, routing the solution to certain reactors, depending on the problem. By storing the intermediate template between 2 valves in the memory, the reactors can be rinsed without interfering with the template.

The sequence of logical operations will determine the settings of the switches. By feeding the problem into a digital computer, the problem will be translated into a sequence of switch settings. The DNA in the microfluidic system will do the actually computing part in a massively parallel fashion, while the electronic part will control the flows by fast switching and monitor the information.

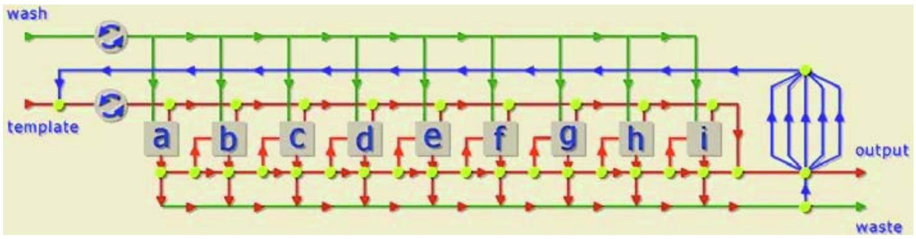


Fig. 3. The architecture of a 9 bit configurable microfluidic computer with 2 pumps (arrows), 9 operators (a-i) and a 5-channel memory. The light grey circles are selection valves

We will illustrate the workings of this system with an example of a NP-complete problem, because this can be readily translated into a set of logic operations.

6 Maximum Clique

The maximum clique problem requires finding the largest subset of fully interconnected nodes in a given graph (Fig. 4). The decision problem associated with the maximum clique problem becomes rapidly harder to solve (NP-complete) as the problem size increases.

The basic algorithm has been discussed by McCaskill [9]. The edges of the graph, *i.e.* the connections between the nodes, can be represented by a so called connectivity matrix [11]. The connectivity matrix for the 9-node example shown in Fig. 4 is the 9x9 matrix in Table 1. As Table 1 shows, the matrix is symmetrical over the diagonal, while the diagonal trivially one, reducing the number of necessary selections from N^2 to ½N(N-1).

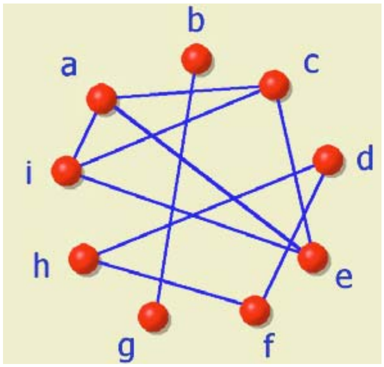


Fig. 4. An N=9 instance of the clique problem. The maximum clique is given by ACEI, represented by 101010001

Table 1. The connectivity matrix for the 9-node graph as shown in Fig. 4. The shaded numbers are trivial selections and don't have to be included in the selection procedure to obtain all the cliques

	A	B	C	D	E	F	G	H	I
A	1	0	1	0	1	0	0	0	1
B	0	1	0	0	0	0	1	0	0
C	1	0	1	0	1	0	0	0	1
D	0	0	0	1	0	1	0	1	0
E	1	0	1	0	1	0	0	0	1
F	0	0	0	1	0	1	0	1	0
G	0	1	0	0	0	0	1	0	0
H	0	0	0	1	0	1	0	1	0
I	1	0	1	0	1	0	0	0	1

The connectivity matrix can be transcribed into a sequence of reactors with the appropriate CPs. Table 2 shows all the CPs necessary to perform the clique selection for the graph in Fig. 4, with instance N=9. If there is a connection, this field stays empty (Ø). For the example presented in Fig. 4, the first column, checking the connectivity with node **A**, would contain CPs labelled with {Ø C_0 Ø E_0 Ø Ø Ø I_0} successively.

The logic operations can be written as follows:

$A_0 \vee [B_k \wedge C_k \wedge D_k \wedge E_k \wedge F_k \wedge G_k \wedge H_k \wedge I_k] \wedge$
$B_0 \vee [C_k \wedge D_k \wedge E_k \wedge F_k \wedge G_k \wedge H_k \wedge I_k] \wedge$
$C_0 \vee [D_k \wedge E_k \wedge F_k \wedge G_k \wedge H_k \wedge I_k] \wedge$
$D_0 \vee [E_k \wedge F_k \wedge G_k \wedge H_k \wedge I_k] \wedge$
$E_0 \vee [F_k \wedge G_k \wedge H_k \wedge I_k] \wedge$
$F_0 \vee [G_k \wedge H_k \wedge I_k] \wedge$
$G_0 \vee [H_k \wedge I_k] \wedge$
$H_0 \vee [I_k]$

with k=0 or Ø

Table 2. The connectivity matrix for the 9-node graph as shown in Fig. 4 transcribed into CPs showing all the necessary selection steps, needed to determine the existence of edges between node i and j. The letter indicates the CP needed for that node while the indices denote the bit value (0 in this case). Ø is an empty selection. The shaded area is programmable and is determined by the edge between node i and j

	node A		node B		node C		node D		E node		node F		node G		node H	
node B	A_0	Ø														
node C	A_0	C_0	B_0	Ø												
node D	A_0	Ø	B_0	Ø	C_0	Ø										
node E	A_0	E_0	B_0	Ø	C_0	E_0	D_0	Ø								
node F	A_0	Ø	B_0	Ø	C_0	Ø	D_0	F_0	E_0	Ø						
node G	A_0	Ø	B_0	G_0	C_0	Ø	D_0	Ø	E_0	Ø	F_0	Ø				
node H	A_0	Ø	B_0	Ø	C_0	Ø	D_0	H_0	E_0	Ø	F_0	H_0	G_0	Ø		
node I	A_0	I_0	B_0	Ø	C_0	I_0	D_0	Ø	E_0	I_0	F_0	Ø	G_0	Ø	H_0	Ø

If there is an edge (k=Ø): $S_{iØ}$ = Ø; no edge (k=0): S_{i0}, with $S_i \in \{A, B, C, \ldots\}$. This results in the following operations for the example given in figure 4:

$$
\begin{aligned}
&A_0 \vee [C_0 \wedge E_0 \wedge I_0] \wedge \\
&B_0 \vee [G_0] \wedge \\
&C_0 \vee [E_0 \wedge I_0] \wedge \\
&D_0 \vee [F_0 \wedge H_0] \wedge \\
&E_0 \vee [I_0] \wedge \\
&F_0 \vee [H_0] \wedge \\
&G_0 \wedge \\
&H_0
\end{aligned}
$$

As we can see from Table 2, some of the CPs are revisited and some are empty. So in principle it is possible to reduce the number of reactors to the number of nodes, represented by the CPs, and revisit the selection modules again if necessary. Since we don't have to check the empty sites, this reduces the number of selection steps from 36x3=108 (as proposed in [9]) to 18 steps, with a maximum of 44 ($\sum_{i=2}^{N} i$) when all nodes are inter-connected. The information flow patterns depend on the switch settings. For node **A**, **B**, **C**, **D** and **H** the flow patterns are presented in Fig. 5a, while a washing step is presented in Fig. 5b.

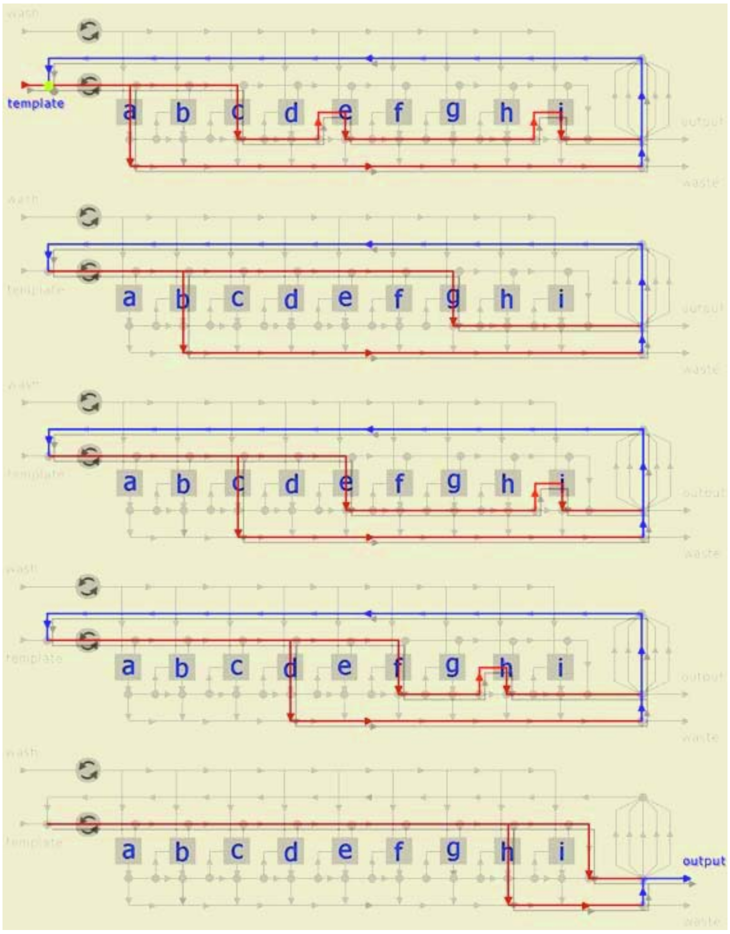


Fig. 5a. The flow pattern for the selections concerning node A, B, C, D and H in the maximum cliques problem

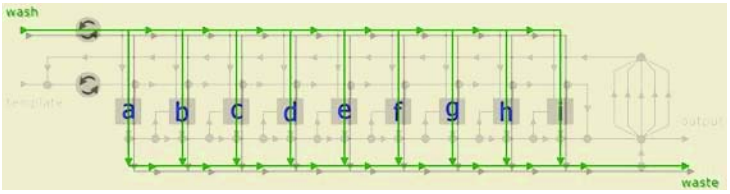


Fig. 5b. The flow pattern for the washing step. Notice that the memory channels are not affected in this step

7 A Simplified 3-Bit Computer

To illustrate the flow patterns, a simple 3-bit programmable computer was implemented in PDMS (Polydimethylsiloxane, Sylgard 184, Dow-Corning, MI). The valves were fabricated on top of the flow channels and were made of PDMS as well [12]. A thin layer of PDMS (~40 µm), which was spin coated, acts like a membrane between a pneumatic actuator and the flow channel. When pressure is applied, the membrane is pushed into the underlying channel, effectively blocking the flow (Fig. 6).

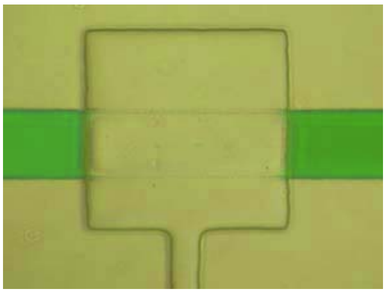


Fig. 6. A fully closed pneumatic valve (300x300 µm square) over a 100 µm channel, effectively blocking the channel

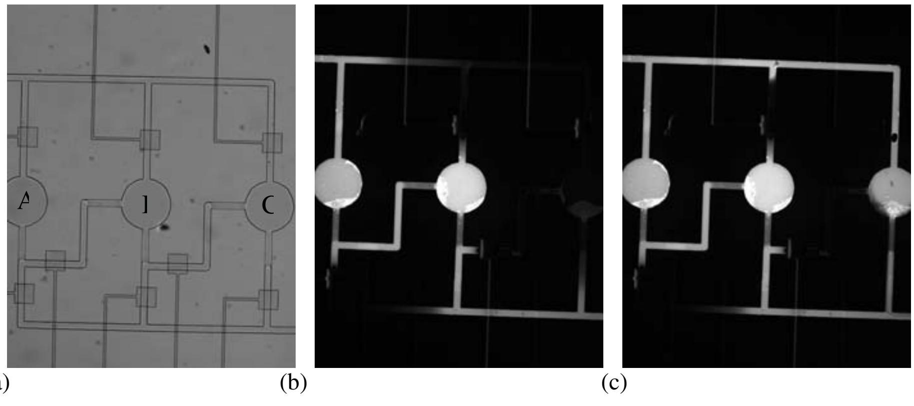


Fig. 7. (a) The lay-out of a simplified 3-bit computer. The circles are the reactors, while the squares are the pneumatic valves. Note that any combination of serial or parallel settings can be made. (b) The reactors A and B are switched in serial: A ∧ B. (c) The same as (b), except reactor C is set in parallel with A and B: (A ∧ B) ∨ C

These valves were pneumatically controlled by a 3-way solenoid valve (Lee Company), which switches the pressure lines between high (30 psi) and low (atmosphere). The solenoid valves where controlled by a conventional computer running LabVIEW (National Instruments). Figure 7a shows the structure of the microsystem. The valves are set up in such fashion that each reactor can be switched into the flow while serial and parallel connections between the reactors are possible. A fluorescence dye was used to highlight various paths in the system. Figure 7b shows the expression A ∧ B while Fig. 7c depicts (A ∧ B) ∨ C.

8 Conclusion

This design is more simple and flexible than previously proposed systems. It further shows that the integration of electronic control with microfluidic is essential for future progress not only in the molecular computing world, but in the biotechnology in general as well. It is easy to see that a system like this can be applied in areas of biotechnology, like in lead optimisation in drug discovery.

Suppose there are a number of diseases to be treated with one optimised drug or *vice versa*, check which disease a drug can treat. The appropriate molecules (DNA, RNA, proteins, *etc.*) could be immobilised in the selection reactors, while the drugs are passed along the reactors. In this manner selections can be made. Even patient specific treatments can be envisioned, by using patient specific data.

It should further be noted that all logic operations can be performed with this system. The end result of the selection has the DNA with the instruction set desired (the DNA is an instruction template).

Acknowledgement

The author wishes to acknowledge the support from DARPA award F30602-01-2-0560 to Laura F. Landweber and NSF award 0121405 to Lydia L. Sohn and Laura F. Landweber. Furthermore the support from the Molecular Evolutionary Computing (MEC) project of the Korean Ministry of Commerce, Industry and Energy, and the National Research Laboratory (NRL) Program from the Korean Ministry of Science and Technology is acknowledged. EEB at Princeton University and the ICT at Seoul National University provided the research facilities.

References

1. van Noort, D. and Austin, R. H. *Towards a bubbling brain in microfluidics.* Manuscript in preparation
2. Hirvensalo, M. (2001) *Quantum computing.* Natural Computing Series, Springer-Verlag, Berlin.
3. Adleman, L. M. (1994) *Molecular computation of solutions to combinatorial problem*s. Science **266**, 1021-1024.
4. Amos, M. and Owenson, G. G. (2000) ERCIM News **43**, 36-37, October 2000.
5. Klein, J. P., Leete, T. H. and Rubin, H. (1999) *A biomolecular implementation of logically reversible computation with minimal energy dissipation.* BioSystems **52**, 15-23.

6. Huang, Y., Duan, X. ,Cui, Y., Lauhon, L., Kim, K. and Lieber, C. M. (2001*) Logic Gates and Computation from Assembled Nanowire Building Blocks.* Science **294**, 1313-1317.
7. Gehani, A. and Reif, J. (1999) *Micro flow bio-molecular computation.* BioSystems **52**, 197-216.
8. McCaskill, J. S. and Wagler, P. (2000) *From reconfigurability to evolution in construction systems: spanning the electronic, microfuidic and biomolecular domains.* **In** R. W. Hartenstein and H. Grünbacher (Eds.) FPL 2000, LNCS **1896**, 286-299, Springer-Verlag, Berlin Heidelberg.
9. McCaskill, J. S. (2001) *Optically programming DNA computing in microflow reactors.* BioSystems **59**, 125-138.
10. van Noort, D and Landweber, L. F. (2003) *Towards a re-programmable DNA computer.* Ninth International Meeting on DNA Based Computers, June 1-4, 2003, USA.
11. van Noort, D., Gast, F.-U. and McCaskill, J. S. (2001) *DNA computing in microreactors.* LNCS 2340, 33-45.
12. Marc A. Unger, Hou-Pu Chou, Todd Thorsen, Axel Scherer, Stephen R. Quake (2000) *Monolithic Microfabricated Valves and Pumps by Multilayer Soft Lithography*, Science **288**, 113-116.

Combinatorial Aspects of Minimal DNA Expressions

Rudy van Vliet, Hendrik Jan Hoogeboom, and Grzegorz Rozenberg

Leiden Institute of Advanced Computer Science (LIACS),
Leiden University,
Niels Bohrweg 1, 2333 CA Leiden, The Netherlands
`{rvvliet, hoogeboo, rozenber}@liacs.nl`

Abstract. We describe a formal language/notation for DNA molecules that may contain nicks and gaps. The elements of the language, DNA expressions, denote formal DNA molecules. Different DNA expressions may denote the same formal DNA molecule. We analyse the shortest DNA expressions denoting a given formal DNA molecule: what is their length, how are they constructed, how many of them are there, and how can they be characterized.

1 Introduction

Since the discovery of the structure and function of DNA molecules, DNA has become an 'intense' research topic among biologists and biochemists. Formal study of computational properties of DNA really began when Head [1987] defined formal languages consisting of strings that can be modified by operations based on the way that restriction enzymes process DNA molecules. Theoretical computer scientists explored the generative power and other properties of such languages, see, e.g., [Kari et al., 1996] and [Head et al., 1997]. The interest of the computer science community in the computational potential of DNA was boosted, when Adleman [1994] described a solution of an instance of the directed Hamiltonian path problem using DNA, enzymes and standard biomolecular operations. Since then, research on DNA computing is really flourishing, see, e.g., [Hagiya & Ohuchi, 2003], [Chen & Reif, 2004] and [Păun et al., 1998]. Recent developments include research on computations in living cells, see, e.g., [Landweber & Kari, 1999], [Daley et al., 2004] and [Ehrenfeucht et al., 2004].

Neither in the theoretical, nor in the applied publications, much attention is paid to the notation used to denote DNA molecules – exceptions are [Boneh et al., 1996] and [Li, 1999]. In most cases, one simply uses the standard double-string notation (like $\begin{matrix}\text{ACATG}\\\text{TGTAC}\end{matrix}$) to describe a double-stranded DNA molecule.

In this paper, we describe a concise and precise notation for DNA molecules, based on the letters A, C, G and T and three operators $\uparrow$, $\downarrow$ and $\updownarrow$ (to be pronounced as *uparrow*, *downarrow* and *updownarrow*, respectively). The resulting DNA expressions denote formal DNA molecules – a formalization of DNA molecules. We do not only account for perfect double-stranded DNA molecules, but also for single-stranded DNA molecules and for double-stranded DNA molecules containing nicks (missing phosphodiester bonds between adjacent nucleotides in the same strand) and gaps (missing nucleotides in one of the strands). The notation is the first step towards a formal description of more complex DNA molecules. The ultimate goal is to also describe the secondary structure of DNA.

Our three operators bear some resemblance to the operators used in [Boneh et al., 1996] and [Li, 1999], but their functionality is quite different. The operator $\uparrow$ acts as a kind of ligase for the upper strands: it creates upper strands and connects the upper strands of its arguments. The operator $\downarrow$ is the analogue for lower strands. Finally, $\updownarrow$ fills up the gap(s) in its argument. The effects of the operators do not perfectly correspond to the effects of existing techniques in real-life DNA synthesis. Yet, the operators are useful to describe certain types of DNA molecules.

In our formal language, different DNA expressions may denote the same formal DNA molecule. We examine which DNA expressions are minimal, i.e., have the shortest length among DNA expressions denoting the same formal DNA molecule, and what their length is. Moreover, there may be different minimal DNA expressions denoting the same formal DNA molecule. We calculate the number of these minimal DNA expressions. Finally, we give a characterization of minimal DNA expressions, which makes it easy to check whether or not a given DNA expression is minimal.

Due to space limitations, we omit the formal proofs of the results we present in this paper. For some results, however, we will provide an intuitive argumentation. The proofs can be found in [Van Vliet, 2004].

2 $\mathcal{N}$-Words and Formal DNA Molecules

The letters A, C, G and T, denoting the four nucleotides that a DNA molecule consists of, are also important building blocks of our language. We use $\mathcal{N}$ to denote this alphabet: $\mathcal{N} = \{\text{A, C, G, T}\}$. The elements of $\mathcal{N}$ are called $\mathcal{N}$-*letters*. A non-empty string over $\mathcal{N}$ is called an $\mathcal{N}$-word.

For an $\mathcal{N}$-word α, $c(\alpha)$ is the element-wise (non-reversed) Watson-Crick *complement* of α. For example, $c(\text{ACATG}) = \text{TGTAC}$.

The semantical basis of our formal language are *formal DNA molecules*. Formal DNA molecules are strings over the set $\mathcal{A}_{\triangledown\vartriangle} = \mathcal{A}_+ \cup \mathcal{A}_- \cup \mathcal{A}_\pm \cup \{^\triangledown, {}_\vartriangle\}$, where $\mathcal{A}_+ = \left\{\binom{\text{A}}{-}, \binom{\text{C}}{-}, \binom{\text{G}}{-}, \binom{\text{T}}{-}\right\}$, $\mathcal{A}_- = \left\{\binom{-}{\text{A}}, \binom{-}{\text{C}}, \binom{-}{\text{G}}, \binom{-}{\text{T}}\right\}$ and $\mathcal{A}_\pm = \left\{\binom{\text{A}}{\text{T}}, \binom{\text{C}}{\text{G}}, \binom{\text{G}}{\text{C}}, \binom{\text{T}}{\text{A}}\right\}$. The elements of $\mathcal{A}_+ \cup \mathcal{A}_- \cup \mathcal{A}_\pm$ are called $\mathcal{A}$-letters. The elements of $\mathcal{A}_+$ and $\mathcal{A}_-$ correspond to gaps in the lower strand and the upper strand, respectively. The symbols $^\triangledown$ and ${}_\vartriangle$ are called *nick letters*. The *upper nick* *letter* $^\triangledown$ represents a nick in the upper strand of the DNA molecule; the *lower nick letter* $_\vartriangle$ represents a nick in the lower strand.

Not all strings over $\mathcal{A}_{\triangledown\vartriangle}$ are formal DNA molecules. We impose three natural conditions on the strings, which, among others, prevent the DNA molecule represented from 'falling apart'.

Definition 1. *A formal DNA molecule is a string $X = x_1x_2\dots x_r$ with $r \geq 1$ and for $i = 1, \dots, r$, $x_i \in \mathcal{A}_{\triangledown\vartriangle}$, satisfying*

- *if $x_i \in \mathcal{A}_+$, then $x_{i+1} \notin \mathcal{A}_-$* $\quad (i = 1, 2, \dots, r-1)$,
  if $x_i \in \mathcal{A}_-$, then $x_{i+1} \notin \mathcal{A}_+$ $\quad (i = 1, 2, \dots, r-1)$,
- $x_1, x_r \notin \{^\triangledown, _\vartriangle\}$,
- *if $x_i \in \{^\triangledown, _\vartriangle\}$, then $x_{i-1}, x_{i+1} \in \mathcal{A}_\pm$* $\quad (i = 2, 3, \dots, r-1)$.

A formal DNA molecule that does not contain nick letters, is called *nick free*.

Examples of formal DNA molecules are

$$X_1 = \binom{\mathrm{A}}{\mathrm{T}}\binom{\mathrm{C}}{\mathrm{G}}\binom{\mathrm{A}}{\mathrm{T}}\binom{\mathrm{T}}{\mathrm{A}}\binom{\mathrm{G}}{\mathrm{C}}, \tag{1}$$

$$X_2 = \binom{\mathrm{A}}{\mathrm{T}}^\triangledown\binom{\mathrm{C}}{\mathrm{G}}\binom{\mathrm{A}}{\mathrm{T}}_\vartriangle\binom{\mathrm{T}}{\mathrm{A}}\binom{\mathrm{G}}{-}, \text{ and} \tag{2}$$

$$X_3 = \binom{-}{\mathrm{T}}\binom{\mathrm{C}}{\mathrm{G}}\binom{\mathrm{A}}{-}\binom{\mathrm{T}}{-}\binom{\mathrm{G}}{\mathrm{C}}. \tag{3}$$

Both X_1 and X_3 are nick free. We assume that if two nucleotides in the same strand are separated by a gap (as is the case for the G and the C in the lower strand of X_3), then they are not connected by a (long) phosphodiester bond.

The following strings over $\mathcal{A}_{\triangledown\vartriangle}$ are no formal DNA molecules, because they violate one of the three conditions from Definition 1.

$$X_1' = \binom{-}{\mathrm{T}}\binom{-}{\mathrm{G}}\binom{\mathrm{A}}{-}\binom{\mathrm{T}}{\mathrm{A}}\binom{\mathrm{G}}{\mathrm{C}},$$

$$X_2' = {}_\vartriangle\binom{\mathrm{A}}{\mathrm{T}}\binom{\mathrm{C}}{\mathrm{G}}\binom{\mathrm{A}}{\mathrm{T}}\binom{\mathrm{T}}{\mathrm{A}}\binom{\mathrm{G}}{-}, \text{ and}$$

$$X_3' = \binom{-}{\mathrm{T}}^\triangledown\binom{\mathrm{C}}{\mathrm{G}}\binom{\mathrm{A}}{-}^\triangledown\binom{\mathrm{T}}{-}\binom{\mathrm{G}}{\mathrm{C}}.$$

Often, we simplify the notation of a formal DNA molecule. Let $X = x_1 \dots x_r$ for some $r \geq 1$ be a formal DNA molecule. If two or more consecutive symbols of X are elements of $\mathcal{A}_+$, say $x_i \dots x_j = \binom{a_i}{-} \dots \binom{a_j}{-}$ with $1 \leq i < j \leq r$, then we may substitute $x_i \dots x_j$ by $\binom{a_i \dots a_j}{-}$. Analogously, we may substitute consecutive elements of $\mathcal{A}_-$ and consecutive elements of $\mathcal{A}_\pm$. When we simplify the *notation* of a formal DNA molecule, we do not modify the formal DNA molecule itself. In particular, it remains a string over $\mathcal{A}_{\triangledown\vartriangle}$.

A non-empty sequence of elements of $\mathcal{A}_+$ is called an *upper $\mathcal{A}$-word*. Analogously, we have a *lower $\mathcal{A}$-word* (with elements of $\mathcal{A}_-$) and a *double $\mathcal{A}$-word* (with elements of $\mathcal{A}_\pm$). These notions are needed to define the decomposition of a formal DNA molecule:

Definition 2. *Let X be a formal DNA molecule. The decomposition of X is the sequence $x'_1, \dots, x'_k$ of $k \geq 1$ non-empty strings over $\mathcal{A}_{\triangledown\vartriangle}$ such that*

- $X = x'_1 \dots x'_k$,
- *for $i = 1, \dots, k$, x'_i is either an upper $\mathcal{A}$-word, or a lower $\mathcal{A}$-word, or a double $\mathcal{A}$-word, or a nick letter, and*
- *for $i = 1, \dots, k-1$, if x'_i is an upper $\mathcal{A}$-word, then x'_{i+1} is not an upper $\mathcal{A}$-word, and analogously for lower $\mathcal{A}$-words and double $\mathcal{A}$-words.*

Hence, the decomposition of X cannot be simplified any further. For the ease of notation, we will in general write $x'_1 \dots x'_k$ instead of $x'_1, \dots, x'_k$.

For example, the decompositions of the formal DNA molecules X_1, X_2 and X_3 are

$$\begin{aligned} X_1 &= \binom{\text{ACATG}}{\text{TGTAC}} && \text{(with } k=1\text{)},\\ X_2 &= \binom{\text{A}}{\text{T}}^{\triangledown}\binom{\text{CA}}{\text{GT}}_{\vartriangle}\binom{\text{T}}{\text{A}}\binom{\text{G}}{-} && \text{(with } k=6\text{), and}\\ X_3 &= \binom{-}{\text{T}}\binom{\text{C}}{\text{G}}\binom{\text{AT}}{-}\binom{\text{G}}{\text{C}} && \text{(with } k=4\text{)}. \end{aligned}$$

If $x'_1 \dots x'_k$ for some $k \geq 1$ is the decomposition of a formal DNA molecule X, then the substrings x'_i are called the *components* of X. For $i = 1, \dots, k$, if x'_i is an upper $\mathcal{A}$-word (lower $\mathcal{A}$-word or double $\mathcal{A}$-word), then it is called an *upper component* (*lower component* or *double component*, respectively) of X. If x'_i is either an upper component or a lower component, then we may also call it a *single-stranded component* of X.

Because, by definition, an upper component of a formal DNA molecule X cannot be followed by a lower component and vice versa, and nick letters occurring in X must be preceded and followed by a double component, we have

Lemma 3. *For each formal DNA molecule X, the decomposition of X is an alternating sequence of double components on the one hand and other types of components on the other hand.*

For example, the decomposition of X_2 consists of a double component, an upper nick letter, a double component, a lower nick letter, a double component and an upper component, respectively.

We define three functions on formal DNA molecules. Let $X = x_1 \dots x_r$ for some $r \geq 1$ be a formal DNA molecule. Then $L(X) = x_1$ and $R(X) = x_r$. Hence, the functions L and R give the leftmost symbol and the rightmost symbol of a formal DNA molecule. Further, $|X|_{\mathcal{A}}$ counts the $\mathcal{A}$-letters occurring in X. For example, $L(X_2) = \binom{\text{A}}{\text{T}}$, $R(X_2) = \binom{\text{G}}{-}$ and $|X_2|_{\mathcal{A}} = 5$.

3 DNA Expressions

The elements of our language are called *DNA expressions*. The semantics of a DNA expression E is a formal DNA molecule, denoted by $\mathcal{S}(E)$. In this paper,

$$\mathcal{S}\left(\left\langle\uparrow \begin{array}{c}\text{C}\\\text{G}\end{array}\ \text{AT}\ \begin{array}{c}\text{G}\overset{\triangledown}{\,}\text{C}\\\text{CG}\end{array}\right\rangle\right) = \begin{array}{l}\text{CATGC}\\\text{G}\quad\text{CG}\end{array} \qquad \mathcal{S}\left(\left\langle\uparrow \begin{array}{c}\text{A}\\\text{T}\end{array}\ \ \begin{array}{c}\text{T}\\\text{A}\end{array}\right\rangle\right) = \begin{array}{c}\text{AT}\\\text{T}\underset{\vartriangle}{\,}\text{A}\end{array} \quad \text{(a)}$$

$$\mathcal{S}\left(\left\langle\downarrow \text{T}\ \begin{array}{c}\text{CATGC}\\\text{G}\quad\text{CG}\end{array}\ \begin{array}{c}\text{AT}\\\text{T}\underset{\vartriangle}{\,}\text{A}\end{array}\right\rangle\right) = \begin{array}{l}\quad\ \text{CATGC}\overset{\triangledown}{\,}\text{AT}\\\text{TG}\quad\text{CGTA}\end{array} \quad \text{(b)}$$

$$\mathcal{S}\left(\left\langle\updownarrow \begin{array}{l}\quad\ \text{CATGC}\overset{\triangledown}{\,}\text{AT}\\\text{TG}\quad\text{CGTA}\end{array}\right\rangle\right) = \begin{array}{l}\text{ACATGC}\overset{\triangledown}{\,}\text{AT}\\\text{TGTACGTA}\end{array} \quad \text{(c)}$$

Fig. 1. Examples of (a) the effect of the operator $\uparrow$; (b) the effect of the operator $\downarrow$; (c) the effect of the operator $\updownarrow$

we will describe the syntax and semantics of a DNA expression in words and by means of examples. For a formal definition, we refer to [Van Vliet, 2004]. One can also define a context-free grammar that generates the DNA expressions.

DNA expressions are the result of applying the three operators $\uparrow$, $\downarrow$ and $\updownarrow$ to basic $\mathcal{N}$-words. In general, the operator $\uparrow$ can have any number $n \geq 1$ arguments $\varepsilon_1, \ldots, \varepsilon_n$, which may be $\mathcal{N}$-words or DNA expressions. The result of applying $\uparrow$ to these arguments is the DNA expression $\langle\uparrow \varepsilon_1 \ldots \varepsilon_n\rangle$. It is called an $\uparrow$-*expression*. Analogously, we may have a $\downarrow$-*expression* $\langle\downarrow \varepsilon_1 \ldots \varepsilon_n\rangle$. The operator $\updownarrow$ can have only one argument ε_1, which may again be an $\mathcal{N}$-word or a DNA expression, yielding an $\updownarrow$-*expression* $\langle\updownarrow \varepsilon_1\rangle$.

Hence, the set of all DNA expressions is a language over the alphabet $\mathcal{N} \cup \{\uparrow, \downarrow, \updownarrow, \langle\ ,\ \rangle\}$. The length of a specific DNA expression E is defined as the number of its symbols and is denoted by $|E|$. The *outermost operator* of a DNA expression is (the occurrence of) the operator which has been performed last. For example, the outermost operator of an $\uparrow$-expression is $\uparrow$. All other occurrences of operators in a DNA expression, i.e., the occurrences in the argument(s) of the outermost operator, are called *inner occurrences*.

The effect of $\uparrow$ is the following: (1) for each argument that is an $\mathcal{N}$-word α, it produces an upper $\mathcal{A}$-word $\binom{\alpha}{-}$, (2) it removes all upper nick letters occurring in its arguments, and (3) it connects the upper strands of consecutive arguments.

Step (3) requires that for $i = 1, \ldots, n-1$, the upper strand of the formal DNA molecule X_i corresponding to argument ε_i extends at least as far to the right as the lower strand: $R(X_i)$ must not be an element of $\mathcal{A}_-$. Analogously, if X_{i+1} is the formal DNA molecule corresponding to argument ε_{i+1}, then $L(X_{i+1})$ must not be an element of $\mathcal{A}_-$. Otherwise, there would be a gap in the upper strand 'between' X_i and X_{i+1}, and we would not be able to connect the upper strands. Such natural requirements are tedious to formalize, which is why we omit a full formal definition of DNA expressions.

Lower nick letters that occur in the arguments of $\uparrow$ are not removed. On the contrary, if both $R(X_i)$ and $L(X_{i+1})$ are elements of $\mathcal{A}_\pm$, then $\uparrow$ produces a lower nick letter between X_i and X_{i+1}. Thus, the lower strands of consecutive arguments are not connected.

The simplest $\uparrow$-expression is of the form $\langle\uparrow \alpha_1\rangle$ for an $\mathcal{N}$-word α_1. Its semantics is the formal DNA molecule $\binom{\alpha_1}{-}$. Figure 1(a) shows the effect of $\uparrow$ for two less trivial examples. For the ease of understanding, we replaced the arguments of $\uparrow$ that are DNA expressions by pictorial representations of the corresponding DNA molecules. The result of the operator is depicted in the same way. For example, the first $\uparrow$-expression has three arguments: a DNA expression, an $\mathcal{N}$-word and another DNA expression, respectively.

On the other hand, although the DNA molecules corresponding to $\begin{smallmatrix}\mathrm{ACAT}\\ \mathrm{TGT}\end{smallmatrix}$ and $\begin{smallmatrix}\mathrm{G}\\ \mathrm{AC}\end{smallmatrix}$ have matching sticky ends, $\left\langle\uparrow\ \begin{smallmatrix}\mathrm{ACAT}\\ \mathrm{TGT}\end{smallmatrix}\ \ \begin{smallmatrix}\mathrm{G}\\ \mathrm{AC}\end{smallmatrix}\right\rangle$ is not a DNA expression, because $L(\binom{-}{\mathrm{A}}\binom{\mathrm{G}}{\mathrm{C}}) \in \mathcal{A}_{-}$. Hence, the operator $\uparrow$ does not account for annealing. Analogously, $\left\langle\uparrow\ \begin{smallmatrix}\mathrm{AC}\\ \mathrm{TGT}\end{smallmatrix}\ \ \begin{smallmatrix}\mathrm{G}\\ \mathrm{AC}\end{smallmatrix}\right\rangle$ is not a DNA expression.

The effect of the operator $\downarrow$ is analogous to that of $\uparrow$. However, instead of upper $\mathcal{A}$-words, upper nick letters and upper strands, we must read lower $\mathcal{A}$-words, lower nick letters and lower strands, respectively. For step (3), we also have analogous requirements. The effect of $\downarrow$ is illustrated in Fig. 1(b).

Finally, the operator $\updownarrow$ complements its argument: it provides a complementary nucleotide for every nucleotide that is not yet complemented. Each nucleotide added is connected to its direct neighbours. The operator does not introduce or remove nick letters. The argument of $\updownarrow$ may be any $\mathcal{N}$-word or any DNA expression. If the argument is an $\mathcal{N}$-word α_1, then it is interpreted as $\langle\uparrow \alpha_1\rangle$. Hence, $\mathcal{S}(\langle\updownarrow \alpha_1\rangle) = \binom{\alpha_1}{c(\alpha_1)}$.

Figure 1(c) illustrates the effect of $\updownarrow$. A complete DNA expression denoting the formal DNA molecule from this example is

$$E = \langle\updownarrow \langle\downarrow \mathrm{T} \langle\uparrow \langle\updownarrow \mathrm{C}\rangle \mathrm{AT} \langle\downarrow \langle\updownarrow \mathrm{G}\rangle \langle\updownarrow \mathrm{C}\rangle\rangle\rangle \langle\uparrow \langle\updownarrow \mathrm{A}\rangle \langle\updownarrow \mathrm{T}\rangle\rangle\rangle\rangle\,. \tag{4}$$

It is the result of the step-by-step construction from Fig. 1.

We say that a formal DNA molecule X is *expressible*, if there exists a DNA expression E with $\mathcal{S}(E) = X$. Unfortunately, there exist formal DNA molecules that are not expressible. In fact, we have:

Theorem 4. *A formal DNA molecule is expressible, if and only if it does not both contain upper nick letters and lower nick letters.*

Hence, the formal DNA molecules X_1 and X_3 from (1) and (3) are expressible, for example by DNA expressions $\langle\updownarrow \mathrm{ACATG}\rangle$ and $\langle\downarrow \mathrm{T} \langle\uparrow \langle\updownarrow \mathrm{C}\rangle \mathrm{AT} \langle\updownarrow \mathrm{G}\rangle\rangle\rangle$, respectively. X_2, however, is not expressible.

4 The Length of a DNA Expression

Different DNA expressions may denote the same formal DNA molecule. Such DNA expressions are called *equivalent*. In fact, for each expressible formal DNA molecule X, there exist infinitely many DNA expressions denoting X. For example, it is easily verified that if E is an $\uparrow$-expression denoting X, then so is $\langle\uparrow E\rangle$. By repeating the construction, adding three symbols (two brackets and an operator) at a time, we can find arbitrarily long, equivalent DNA expressions. Hence, there is no maximal length for DNA expressions denoting a given formal

DNA molecule. There does, however, exist a minimal length. We will examine this length for all types of expressible formal DNA molecules and we will also describe the DNA expressions that achieve this length. Before we do so, we make an elementary observation:

Lemma 5. *Let E be a DNA expression denoting a formal DNA molecule X, and let p be the number of operators occurring in E. Then $|E| = 3 \cdot p + |X|_{\mathcal{A}}$.*

Because each occurrence of an operator is accompanied by an opening bracket and a closing bracket, the term $3 \cdot p$ accounts for the operators and the brackets in E. Consequently, $|X|_{\mathcal{A}}$ counts the $\mathcal{N}$-letters occurring in E. Note that this number only depends on X, and not on the specific DNA expression E.

Indeed, for the DNA expression E from (4), the number p of operators is 10, the number of $\mathcal{A}$-letters in X is 8 and $|E| = 3 \cdot 10 + 8 = 38$.

5 Lower Bounds for the Length of a DNA Expression

We first examine lower bounds for the length of a DNA expression E denoting a formal DNA molecule X. These lower bounds will be expressed in terms of some simple counting functions of X. We now introduce these counting functions.

It follows from the definition of a DNA expression that both *upper* components and *lower* nick letters are the result of an occurrence of the operator $\uparrow$. Therefore, these type of components are called $\uparrow$*-components*. Analogously, lower components and upper nick letters are called $\downarrow$*-components*.

Recall that the decomposition of a formal DNA molecule is an alternating sequence of double components on the one hand and other types of components on the other hand. If we disregard the double components, then we only have a sequence of other types of components, which are $\uparrow$-components and $\downarrow$-components. Consecutive $\uparrow$-components form a *(maximal) series* of $\uparrow$-components. Analogously, we have maximal series of $\downarrow$-components.

Definition 6. *Let X be a formal DNA molecule.*

- *$T_{\uparrow}(X)$ is the number of maximal series of $\uparrow$-components of X.*
- *$T_{\downarrow}(X)$ is the number of maximal series of $\downarrow$-components of X.*
- *$n_{\updownarrow}(X)$ is the number of double components of X.*

We illustrate this definition by the formal DNA molecule X depicted in Fig. 2. The α_i's occurring in this picture denote the $\mathcal{N}$-words determining the upper, lower and double components of X. The $\uparrow$-components of X are $\binom{\alpha_4}{-}$ (series 1), $\binom{\alpha_8}{-}$ and $\binom{\alpha_{10}}{-}$ (series 2), and $\binom{\alpha_{13}}{-}$ (series 3). The $\downarrow$-components of X are the first and the second upper nick letter (series 1), $\binom{-}{\alpha_6}$ (series 2), the third upper nick letter (series 3) and the fourth upper nick letter and $\binom{-}{\alpha_{16}}$ (series 4). Hence, $T_{\uparrow}(X) = 3$ and $T_{\downarrow}(X) = 4$. Further, $n_{\updownarrow}(X) = 10$.

Intuitively, $T_{\uparrow}(X)$ counts the *transitions* (from $\downarrow$-components) to $\uparrow$-components. It requires an occurrence of the operator $\uparrow$ to achieve this transition.

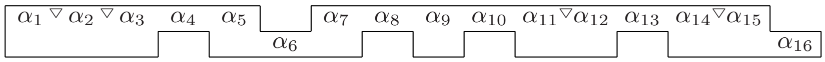

$\alpha_1 \triangledown \alpha_2 \triangledown \alpha_3$
α_4
α_5
α_6
α_7
α_8
α_9
α_{10}
$\alpha_{11} \triangledown \alpha_{12}$
α_{13}
$\alpha_{14} \triangledown \alpha_{15}$
α_{16}

Fig. 2. Pictorial representation of a formal DNA molecule containing upper nick letters

There is, of course, an analogous interpretation of $T_{\downarrow}(X)$. Note that, unless a formal DNA molecule X only consists of a double component, hence, unless $X = \binom{\alpha_1}{c(\alpha_1)}$ for an $\mathcal{N}$-word α_1, either $T_{\uparrow}(X) > 0$, or $T_{\downarrow}(X) > 0$ (or both).

Because maximal series of $\uparrow$-components and maximal series of $\downarrow$-components alternate in a formal DNA molecule, we have

Lemma 7. *For each formal DNA molecule X, $T_{\uparrow}(X)-1 \leq T_{\downarrow}(X) \leq T_{\uparrow}(X)+1$.*

We can now formulate lower bounds on the lengths of DNA expressions:

Theorem 8. *Let E be a DNA expression, and let $X = \mathcal{S}(E)$.*

1. *If E is an $\uparrow$-expression, then $|E| \geq 3 + 3 \cdot T_{\downarrow}(X) + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}}$.*
2. *If E is a $\downarrow$-expression, then $|E| \geq 3 + 3 \cdot T_{\uparrow}(X) + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}}$.*
3. *If $E = \langle \updownarrow \alpha_1 \rangle$ for an $\mathcal{N}$-word α_1, then $|E| = 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}}$.*
4. *If $E = \langle \updownarrow E_1 \rangle$ for a DNA expression E_1, then $|E| \geq 3 + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}}$.*

The terms $3+3 \cdot T_{\downarrow}(X)$ and $3+3 \cdot T_{\uparrow}(X)$ occurring in the first two lower bounds correspond to occurrences of the two operators $\uparrow$ and $\downarrow$. The term $3 \cdot n_{\updownarrow}(X)$ occurring in all lower bounds corresponds to occurrences of the operator $\updownarrow$, which are needed to obtain the double components of X. For example, an $\uparrow$-expression denoting a formal DNA molecule X contains at least $(1 + T_{\downarrow}(X))$ occurrences of $\uparrow$ and $\downarrow$ together, and at least $n_{\updownarrow}(X)$ occurrences of $\updownarrow$.

The symmetry between Claims 1 and 2 is due to the symmetrical effects of the operators $\uparrow$ and $\downarrow$. In later results, we will refer to this symmetry rather than fully stating a symmetrical claim.

6 Minimal DNA Expressions for a Nick Free Molecule

We are not just interested in lower bounds on the lengths of DNA expressions; we also want to be able to *construct* the shortest DNA expressions denoting a given formal DNA molecule. A DNA expression E is called *minimal*, if for every equivalent DNA expression E', $|E'| \geq |E|$.

We first consider nick free formal DNA molecules. These consist only of upper components, lower components and double components. By Theorem 4, each nick free formal DNA molecule is expressible.

Theorem 9. *Let X be a nick free formal DNA molecule.*

1. *If $X = \binom{\alpha_1}{c(\alpha_1)}$ for an $\mathcal{N}$-word α_1, then the only minimal DNA expression denoting X is $E = \langle \updownarrow \alpha_1 \rangle$, with length $|E| = 3 + |X|_{\mathcal{A}}$.*
2. *If $T_{\uparrow}(X) = T_{\downarrow}(X) \geq 1$, then each minimal DNA expression E denoting X is either an $\uparrow$-expression or a $\downarrow$-expression and has length*

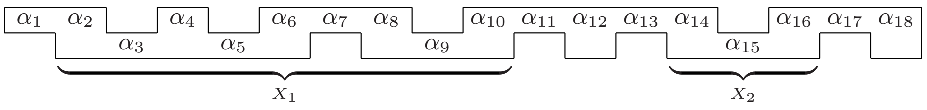


Fig. 3. Pictorial representation of a nick free formal DNA molecule X with single-stranded components. The lower components have been partitioned in submolecules X_1 and X_2 (see the construction below Theorem 9)

$$\begin{aligned}|E| &= 3 + 3 \cdot T_{\downarrow}(X) + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}} \\ &= 3 + 3 \cdot T_{\uparrow}(X) + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}}.\end{aligned}$$

3. *If $T_{\uparrow}(X) > T_{\downarrow}(X)$, then each minimal DNA expression E denoting X is an $\uparrow$-expression and has length*

$$|E| = 3 + 3 \cdot T_{\downarrow}(X) + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}}.$$

4. *If $T_{\downarrow}(X) > T_{\uparrow}(X)$, then ... (symmetric to Claim 3).*

For nick free formal DNA molecules with at least one single-stranded component, we did not mention *how to construct* the minimal DNA expressions. We will describe this construction now, in an intuitive way, by means of an example.

Consider the nick free formal DNA molecule X depicted in Fig. 3, for which $T_{\uparrow}(X) = 4$, $T_{\downarrow}(X) = 3$ and $n_{\updownarrow}(X) = 9$. Because $T_{\uparrow}(X) > T_{\downarrow}(X)$, a minimal DNA expression denoting X must be an $\uparrow$-expression. Upper components $\binom{\alpha_i}{-}$ result when $\uparrow$ has arguments that are $\mathcal{N}$-words α_i, and double components $\binom{\alpha_i}{c(\alpha_i)}$ of X can be produced efficiently by arguments of the form $\langle \updownarrow \alpha_i \rangle$. The lower components of X, however, require a special treatment. We partition the lower components of X in submolecules $X_1, X_2, \ldots, X_r$ for some $r \geq 1$, which, if possible, start with a double component preceding a maximal series of $\downarrow$-components and end with a double component succeeding a maximal series of $\downarrow$-components. If the first component of X is a $\downarrow$-component, then X_1 starts with this component. Analogously, X_r may end with a $\downarrow$-component.

For our nick free formal DNA molecule X, we may take $r = 2$ and

$$\begin{aligned}X_1 &= \binom{\alpha_2}{c(\alpha_2)}\binom{-}{\alpha_3}\binom{\alpha_4}{c(\alpha_4)}\binom{-}{\alpha_5}\binom{\alpha_6}{c(\alpha_6)}\binom{\alpha_7}{-}\binom{\alpha_8}{c(\alpha_8)}\binom{-}{\alpha_9}\binom{\alpha_{10}}{c(\alpha_{10})}, \\ X_2 &= \binom{\alpha_{14}}{c(\alpha_{14})}\binom{-}{\alpha_{15}}\binom{\alpha_{16}}{c(\alpha_{16})}\end{aligned}$$

(see also Fig. 3). We now recursively determine minimal DNA expressions E_1 and E_2 denoting X_1 and X_2, respectively. These minimal DNA expressions become arguments of the $\uparrow$-expression E we are constructing, together with $\mathcal{N}$-words α_i and $\updownarrow$-expressions $\langle \updownarrow \alpha_i \rangle$ for the upper components and double components of X which are neither in X_1, nor in X_2:

$$E = \langle \uparrow \alpha_1 E_1 \alpha_{11} \langle \updownarrow \alpha_{12} \rangle \alpha_{13} E_2 \alpha_{17} \langle \updownarrow \alpha_{18} \rangle \rangle.$$

Because $T_{\uparrow}(X_1) = 1 < 2 = T_{\downarrow}(X_1)$, E_1 must be a $\downarrow$-expression, which is constructed in an analogous way: the only upper component of X_1 is 'partitioned' in the submolecule $X_{1,1} = \binom{\alpha_6}{c(\alpha_6)}\binom{\alpha_7}{-}\binom{\alpha_8}{c(\alpha_8)}$. We determine a minimal DNA expression $E_{1,1}$ for $X_{1,1}$, which is, in turn, an $\uparrow$-expression:

$$E_{1,1} = \langle \uparrow \langle \updownarrow \alpha_6 \rangle \, \alpha_7 \, \langle \updownarrow \alpha_8 \rangle \rangle \, .$$

We then get

$$E_1 = \langle \downarrow \langle \updownarrow \alpha_2 \rangle \, \alpha_3 \, \langle \updownarrow \alpha_4 \rangle \, \alpha_5 \, \langle \uparrow \langle \updownarrow \alpha_6 \rangle \, \alpha_7 \, \langle \updownarrow \alpha_8 \rangle \rangle \, \alpha_9 \, \langle \updownarrow \alpha_{10} \rangle \rangle \, .$$

The minimal DNA expression E_2 is relatively easy to construct:

$$E_2 = \langle \downarrow \langle \updownarrow \alpha_{14} \rangle \, \alpha_{15} \, \langle \updownarrow \alpha_{16} \rangle \rangle \, .$$

Consequently,

$$\begin{aligned} E = \langle \uparrow \quad & \alpha_1 \quad \langle \downarrow \langle \updownarrow \alpha_2 \rangle \, \alpha_3 \, \langle \updownarrow \alpha_4 \rangle \, \alpha_5 \, \langle \uparrow \langle \updownarrow \alpha_6 \rangle \, \alpha_7 \, \langle \updownarrow \alpha_8 \rangle \rangle \, \alpha_9 \, \langle \updownarrow \alpha_{10} \rangle \rangle \\ & \alpha_{11} \, \langle \updownarrow \alpha_{12} \rangle \, \alpha_{13} \quad \langle \downarrow \langle \updownarrow \alpha_{14} \rangle \, \alpha_{15} \, \langle \updownarrow \alpha_{16} \rangle \rangle \quad \alpha_{17} \, \langle \updownarrow \alpha_{18} \rangle \quad \rangle \, . \end{aligned} \tag{5}$$

Indeed,

$$|E| = 39 + |X|_{\mathcal{A}} = 3 + 3 \cdot T_{\downarrow}(X) + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}}.$$

All minimal DNA expressions denoting X are constructed in this way. The result only depends on the way that lower components (for a minimal $\uparrow$-expression) or upper components (for a minimal $\downarrow$-expression) are partitioned in submolecules $X_1, \ldots, X_r$. The construction (of one minimal DNA expression) takes a time linear in the length of X.

The construction of a minimal $\uparrow$-expression for a formal DNA molecule X with $T_{\uparrow}(X) = T_{\downarrow}(X) \geq 1$ (see Theorem 9(2)) proceeds along the same lines.

7 Minimal DNA Expressions for a Molecule with Nicks

To construct minimal DNA expressions for an expressible formal DNA molecule X containing nick letters, we first decompose X into nick free pieces and nick letters. We call the result the *nick free decomposition* of X.

Consider, for example, the formal DNA molecule X depicted in Fig. 4. This molecule contains three lower nick letters and no upper nick letters. The nick free decomposition of X is $Z_1 {}_\triangle Z_2 {}_\triangle Z_3 {}_\triangle Z_4$, where

$$\begin{aligned} Z_1 &= \binom{\alpha_1}{-}\binom{\alpha_2}{c(\alpha_2)}\binom{-}{\alpha_3}\binom{\alpha_4}{c(\alpha_4)}, \\ Z_2 &= \binom{\alpha_5}{c(\alpha_5)}\binom{-}{\alpha_6}\binom{\alpha_7}{c(\alpha_7)}\binom{\alpha_8}{-}\binom{\alpha_9}{c(\alpha_9)}\binom{-}{\alpha_{10}}\binom{\alpha_{11}}{c(\alpha_{11})}, \\ Z_3 &= \binom{\alpha_{12}}{c(\alpha_{12})}\binom{\alpha_{13}}{-}\binom{\alpha_{14}}{c(\alpha_{14})}\binom{\alpha_{15}}{-}\binom{\alpha_{16}}{c(\alpha_{16})}, \\ Z_4 &= \binom{\alpha_{17}}{c(\alpha_{17})}. \end{aligned}$$

A DNA expression E is called *operator-minimal*, if for every equivalent DNA expression E' with the same outermost operator, $|E'| \geq |E|$. For example, consider the formal DNA molecule Z_2, for which $T_{\uparrow}(Z_2) = 1$, $T_{\downarrow}(Z_2) = 2$ and $n_{\updownarrow}(Z_2) = 4$. The $\uparrow$-expression

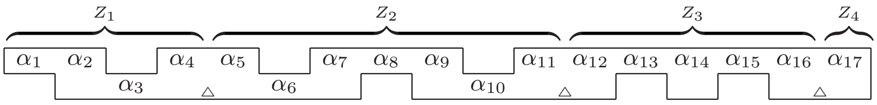

Z_1
Z_2
Z_3
Z_4
α_1 α_2 α_4 α_5 α_7 α_8 α_9 α_{11} α_{12} α_{13} α_{14} α_{15} α_{16} α_{17}
α_3 $\triangle$ α_6 α_{10} $\triangle$ $\triangle$

Fig. 4. Pictorial representation of a formal DNA molecule X containing lower nick letters. The nick free decomposition of X is $Z_1 {}_\triangle Z_2 {}_\triangle Z_3 {}_\triangle Z_4$

$$E_2 = \langle \uparrow \langle \downarrow \langle \updownarrow \alpha_5 \rangle \, \alpha_6 \, \langle \updownarrow \alpha_7 \rangle \rangle \, \alpha_8 \, \langle \downarrow \langle \updownarrow \alpha_9 \rangle \, \alpha_{10} \, \langle \updownarrow \alpha_{11} \rangle \rangle \rangle \, ,$$

which denotes Z_2 and has length

$$|E_2| = 21 + |Z_2|_{\mathcal{A}} = 3 + 3 \cdot T_\downarrow(Z_2) + 3 \cdot n_\updownarrow(Z_2) + |Z_2|_{\mathcal{A}},$$

is operator-minimal, because by Theorem 8(1), there can be no shorter $\uparrow$-expression denoting Z_2.

However, because $T_\downarrow(Z_2) > T_\uparrow(Z_2)$, E_2 is not minimal. By Theorem 9(4), each minimal DNA expression E_2' denoting Z_2 is a $\downarrow$-expression and has length

$$|E_2'| = 3 + 3 \cdot T_\uparrow(Z_2) + 3 \cdot n_\updownarrow(Z_2) + |Z_2|_{\mathcal{A}} = 18 + |Z_2|_{\mathcal{A}}.$$

We are in particular interested in operator-minimal $\uparrow$-expressions and $\downarrow$-expressions denoting nick free formal DNA molecules. These operator-minimal DNA expressions appear to be constructed in exactly the same way as the minimal $\uparrow$-expressions and $\downarrow$-expressions for nick free formal DNA molecules, which we have seen in the previous section. The only difference is that operator-minimal $\uparrow$-expressions and $\downarrow$-expressions can be constructed for *every* nick free formal DNA molecule, and not just for formal DNA molecules X satisfying certain conditions on $T_\uparrow(X)$ and $T_\downarrow(X)$.

We can now describe the minimal DNA expressions denoting expressible formal DNA molecules containing nick letters. We only give the formulation for molecules with lower nick letters, as the formulation for the case with upper nick letters is completely analogous. Note that by definition, there do not exist $\downarrow$-expressions that denote a formal DNA molecule containing lower nick letters.

Theorem 10. *Let X be an expressible formal DNA molecule which contains at least one lower nick letter $\triangle$, and let $Z_1 {}_\triangle Z_2 {}_\triangle \ldots {}_\triangle Z_m$ for some $m \geq 2$ be the nick free decomposition of X.*

For $h = 1, \ldots, m$, let E_h be an operator-minimal $\uparrow$-expression denoting Z_h and let the string $\widehat{E}_h$ be the sequence of the arguments of E_h. Then $E = \left\langle \uparrow \widehat{E}_1 \ldots \widehat{E}_m \right\rangle$ is a minimal DNA expression denoting X and

$$|E| = 3 + 3 \cdot T_\downarrow(X) + 3 \cdot n_\updownarrow(X) + |X|_{\mathcal{A}}.$$

Each minimal DNA expression denoting X is constructed in this way.

We return to the formal DNA molecule X from Fig. 4, for which $T_\downarrow(X) = 3$ and $n_\updownarrow(X) = 10$. We already established the nick free decomposition $Z_1 {}_\triangle Z_2 {}_\triangle Z_3 {}_\triangle Z_4$ of X and considered an operator-minimal $\uparrow$-expression E_2 denoting Z_2. It is not difficult to also construct operator-minimal $\uparrow$-expressions for Z_1, Z_3 and Z_4:

$$\begin{aligned} E_1 &= \langle \uparrow \alpha_1 \langle \downarrow \langle \updownarrow \alpha_2 \rangle \alpha_3 \langle \updownarrow \alpha_4 \rangle \rangle \rangle \, , \\ E_3 &= \langle \uparrow \langle \updownarrow \alpha_{12} \rangle \alpha_{13} \langle \updownarrow \alpha_{14} \rangle \alpha_{15} \langle \updownarrow \alpha_{16} \rangle \rangle \, , \\ E_4 &= \langle \uparrow \langle \updownarrow \alpha_{17} \rangle \rangle \, . \end{aligned}$$

The corresponding minimal DNA expression denoting the entire formal DNA molecule X is

$$\begin{aligned} E = \langle \uparrow \quad & \alpha_1 \langle \downarrow \langle \updownarrow \alpha_2 \rangle \alpha_3 \langle \updownarrow \alpha_4 \rangle \rangle \quad \langle \downarrow \langle \updownarrow \alpha_5 \rangle \alpha_6 \langle \updownarrow \alpha_7 \rangle \rangle \alpha_8 \langle \downarrow \langle \updownarrow \alpha_9 \rangle \alpha_{10} \langle \updownarrow \alpha_{11} \rangle \rangle \\ & \langle \updownarrow \alpha_{12} \rangle \alpha_{13} \langle \updownarrow \alpha_{14} \rangle \alpha_{15} \langle \updownarrow \alpha_{16} \rangle \quad \langle \updownarrow \alpha_{17} \rangle \quad \rangle \, . \end{aligned}$$

Indeed,

$$|E| = 42 + |X|_{\mathcal{A}} = 3 + 3 \cdot T_{\downarrow}(X) + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}}.$$

Also this construction requires linear time.

8 The Number of Minimal DNA Expressions

In principle, there may be many different minimal DNA expressions which denote the same formal DNA molecule. This is due to the different partitionings of lower or upper components that we can choose for the construction of an (operator-)-minimal $\uparrow$-expression or $\downarrow$-expression denoting a nick free formal DNA molecule.

Let X be a nick free formal DNA molecule. There appears to be an elegant bijection between (operator-)minimal $\uparrow$-expressions E denoting X and sequences of $T_{\downarrow}(X)$ well-nested pairs of brackets. This sequence is obtained from E by removing all symbols from E except the brackets corresponding to inner occurrences of the operators $\uparrow$ and $\downarrow$. The result for the minimal $\uparrow$-expression from (5) is $\langle\langle\rangle\rangle\,\langle\rangle$, which is indeed a sequence of $T_{\downarrow}(X) = 3$ well-nested pairs of brackets.

The number of such sequences is one of the many combinatorial interpretations of the well-known Catalan numbers (see [Stanley, 1999]). For $p \geq 0$, there exist $C_p = \frac{1}{p+1}\binom{2p}{p}$ sequences of p well-nested pairs of brackets.

Now, for an expressible formal DNA molecule X, let $n_{\min}(X)$ be the number of different minimal DNA expressions denoting X. We have:

Theorem 11. *Let X be an expressible formal DNA molecule.*

1. *If X is $\langle \updownarrow \alpha_1 \rangle$ for an $\mathcal{N}$-word α_1, then $n_{min}(X) = 1$.*
2. *If X is nick free and $T_{\uparrow}(X) = T_{\downarrow}(X) = p$ with $p \geq 1$, then $n_{min}(X) = 2 \cdot C_p$.*
3. *If X is nick free and $T_{\uparrow}(X) > T_{\downarrow}(X) = p$ with $p \geq 0$, then $n_{min}(X) = C_p$.*
4. *If X is nick free and $T_{\downarrow}(X) > T_{\uparrow}(X) = p$ with $p \geq 0$, then ... (symmetric to Claim 3).*
5. *If X contains at least one lower nick letter, then let $Z_1{}_{\vartriangle}Z_2{}_{\vartriangle} \cdots {}_{\vartriangle}Z_m$ for some $m \geq 2$ be the nick free decomposition of X, and let for $h = 1, \ldots, m$, $p_h = T_{\downarrow}(Z_h)$. Then $n_{min}(X) = C_{p_1} \times \cdots \times C_{p_m}$.*
6. *If X contains at least one upper nick letter, then ... (symmetric to Claim 5).*

9 Characterization of Minimal DNA Expressions

When we want to decide whether or not a given DNA expression E is minimal, we can determine its semantics $X = \mathcal{S}(E)$, look up the length of a minimal DNA expression denoting X and compare this to the length $|E|$ of E. There is, however, also a direct way, based on the following characterization:

Theorem 12. *A DNA expression E is minimal, if and only if*

- *each occurrence of the operator $\updownarrow$ in E has as its argument an $\mathcal{N}$-word α (i.e., not a DNA expression),*
- *and no occurrence of the operator $\uparrow$ in E has an argument that is an $\uparrow$-expression, and no occurrence of the operator $\downarrow$ in E has an argument that is a $\downarrow$-expression,*
- *and unless $E = \langle\uparrow \alpha\rangle$ or $E = \langle\downarrow \alpha\rangle$ for an $\mathcal{N}$-word α, each occurrence of an operator $\uparrow$ or $\downarrow$ in E has at least two arguments,*
- *and for each inner occurrence of an operator $\uparrow$ or $\downarrow$ in E, the arguments are $\mathcal{N}$-words and DNA expressions, alternately,*
- *and for each inner occurrence of an operator $\uparrow$ or $\downarrow$ in E,*
  - *the first argument is either an $\mathcal{N}$-word or an $\updownarrow$-expression,*
  - *and the last argument is either an $\mathcal{N}$-word or an $\updownarrow$-expression,*
- *and if the outermost operator of E is $\uparrow$ or $\downarrow$, then*
  - *either it has two consecutive arguments which are DNA expressions,*
  - *or its first argument is an $\mathcal{N}$-word or an $\updownarrow$-expression,*
  - *or its last argument is an $\mathcal{N}$-word or an $\updownarrow$-expression.*

For an arbitrary DNA expression E, we can verify these six properties in a time linear in the length of E.

10 Conclusions and Directions for Future Research

We have introduced DNA expressions as a formal notation for DNA molecules that may contain nicks and gaps. There exist, however, (formal) DNA molecules with nicks that cannot be represented. For each expressible formal DNA molecule, we have described the minimal DNA expression(s) denoting it and we have determined the number of such minimal DNA expressions. For almost all types of expressible formal DNA molecules, the number of minimal DNA expressions can be expressed in terms of the Catalan numbers. Finally, we have characterized minimal DNA expressions by six properties which can easily be verified.

Because each expressible formal DNA molecule can be denoted by infinitely many DNA expressions, one may ask for a normal form: a well-defined set of properties such that for each expressible formal DNA molecule X, there is a unique DNA expression denoting X and satisfying those properties. And given a normal form, one may ask for an algorithm that, for each DNA expression, determines the equivalent DNA expression in normal form. We already have a normal form and a corresponding algorithm for nick free formal DNA molecules.

We also have ideas for another normal form and a corresponding algorithm, which applies to *all* expressible formal DNA molecules. A nice feature of this new normal form is that each DNA expression satisfying it is minimal.

One may also consider a new set of operators to construct DNA expressions. The result may be that each formal DNA molecule becomes expressible, or that two formal DNA molecules with complementary sticky ends can anneal. It would be desirable/interesting to find extensions such that the new DNA expressions could be used to denote DNA molecules with a variety of other 'imperfections', such as, e.g., hairpin loops and circular strands.

References

L.M. Adleman: Molecular computation of solutions to combinatorial problems, *Science* **266** (1994), 1021-1024.

D. Boneh, C. Dunworth, R.J. Lipton: Breaking DES using a molecular computer, *DNA based computers – Proceedings of a DIMACS workshop* (R.J. Lipton, E.B. Baum, eds.), American Mathematical Society, Providence, RI (1996), 37-66.

J. Chen, J. Reif (eds.): *DNA computing – 9th International workshop on DNA based computers*, LNCS **2943**, Springer-Verlag, Berlin (2004).

M. Daley, L. Kari, I. McQuillan: Families of languages defined by ciliate bio-operations, *Theoretical Computer Science* **320**(1) (2004), 51-69.

A. Ehrenfeucht, T. Harju, I. Petre, D.M. Prescott, G. Rozenberg: *Computation in living cells – Gene assembly in ciliates*, Springer-Verlag, Berlin (2004).

M. Hagiya, A. Ohuchi (eds.): *DNA computing – 8th International workshop on DNA-based computers*, LNCS **2568**, Springer-Verlag, Berlin (2003).

T. Head: Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviors, *Bulletin of Mathematical Biology* **49**(6) (1987), 737-759.

T. Head, Gh. Păun, D. Pixton: Language theory and molecular genetics: generative mechanisms suggested by DNA recombination, *Handbook of formal languages* (G. Rozenberg, A. Salomaa, eds.), Vol. 2, Springer-Verlag, Berlin (1997), 295-360.

L. Kari, Gh. Păun, A. Salomaa: The power of restricted splicing with rules from a regular language, *Journal of Universal Computer Science* **2**(4) (1996), 224-240.

L.F. Landweber, L. Kari: The evolution of cellular computing: nature's solution to a computational problem, *Proceedings of the fourth international meeting on DNA based computers*, *BioSystems* **52** (1999), 3-13.

Z. Li: Algebraic properties of DNA operations, *Proceedings of the fourth international meeting on DNA based computers*, *BioSystems* **52** (1999), 55-61.

Gh. Păun, G. Rozenberg, A. Salomaa: *DNA computing – New computing paradigms*, Springer-Verlag, Berlin (1998).

R. P. Stanley: *Enumerative combinatorics*, Vol. 2, Cambridge University Press, Cambridge (1999).

R. van Vliet: Combinatorial aspects of minimal DNA expressions (ext.), Technical Report 2004-03, Leiden Institute of Advanced Computer Science, Leiden University (2004), see `www.liacs.nl/home/rvvliet/mindnaexpr.html`.

A Design for Cellular Evolutionary Computation by Using Bacteria

Kenichi Wakabayashi and Masayuki Yamamura

Interdisciplinary Graduate School of Science and Engineering,
Tokyo Institute of Technology, 4259 Nagatsuta, Yokohama 226-8502, Japan
wakabayashi@es.dis.titech.ac.jp,
http://www.es.dis.titech.ac.jp

Abstract. In this paper, we propose a general idea of Cellular Evolutionary Computation (CEC). CEC is Evolutionary Computation that solves the optimization problems with real DNA molecules and cells. The easiest means of cellular evolution is achieved by adding some genes to the main frame of gene network in the cell. However, in some cases it is necessary to optimize the gene parameters to achieve a desirable gene network output. We are working toward a realization of Evolutionary Computation algorithm to deal with the network optimization problems. We also suggest a novel method to realize a crossover operator for CEC via homologous recombination system within bacterial cells. Our ultimate objective of this study is the achievement of gene network evolution of the cell. We suggest an idea of cell-based computing that the cell-related problems are addressed by their related cells.

1 Introduction

Recently, various DNA-based computational models were proposed. GA-like approaches were also developped [1, 2, 3]. *In vitro* molecular evolution that is also considered as a computation that mainly targetted at evolution of peptides, RNA, and DNA is commonly studied in biotechnology. A typical scheme of molecular evolution is comprised of several steps - a step of generating mutants, evaluating phenotype of each mutant, and selecting an elite among a population.

The first step is designed to generate a variety to the nucleotide sequences. How to design this step determines the search ability of the system. The traditional approach is classified into a local search that is based on generating several single-base substitutions in the polynucleotides. The crossover methods including DNA Shuffling provide greater variety to a population [4]. Block Shuffling provides diversified library that complies with global search requirements like Genetic Algorithm (GA) [5]. In the steps of evaluation and selection, activities of molecules are rated and we sort an elite through thousands of molecules. As for the peptide molecules, we need to know the corresponding DNA sequence of selected molecule. Therefore, various display methods are developed to create peptide-nucleic acid complex [6, 7, 8].

There are various works and some of them appear to be succeeding in actually progressing molecular activity. Some works modeled an operation after evolutionary computation *in sillico* [9].

In contrast, research of directed cellular evolution is less commonly studied. Recent work by Yokobayashi et al. described about directed evolution of genetic circuit in the cellular environment [10]. In the cell, a vast array of genes and their products keep interactions with each other and with other substances [11]. The interactions between genes and molecules form a huge network. When trying to add new genes into the cell, we need to devote attention to the interactions between new genes and the preexisting Gene Network. Those newly introduced genes sometimes have undesirable effects to the Gene Network. The same applies to the Gene Network that negatively affects the newly introduced genes. Since the effects are sometimes serious and it is not always easy to predict, we should optimize gene expressions by evolutionary computational approach. Cellular evolution is achieved by the adaptive strategy.

2 Design of Cellular Evolutionary Computation

The approach of CEC is simillar to Genetic Algorithm on some level (Figure 1). At first, a plasmid library with a large number of plasmids is constructed (initial population). The plasmids are introduced into bacterial cells. In the cell, the plasmids exchange their information by homologous recombination reaction processed by the host cell (crossover). The cells are spread on an agarose plate.

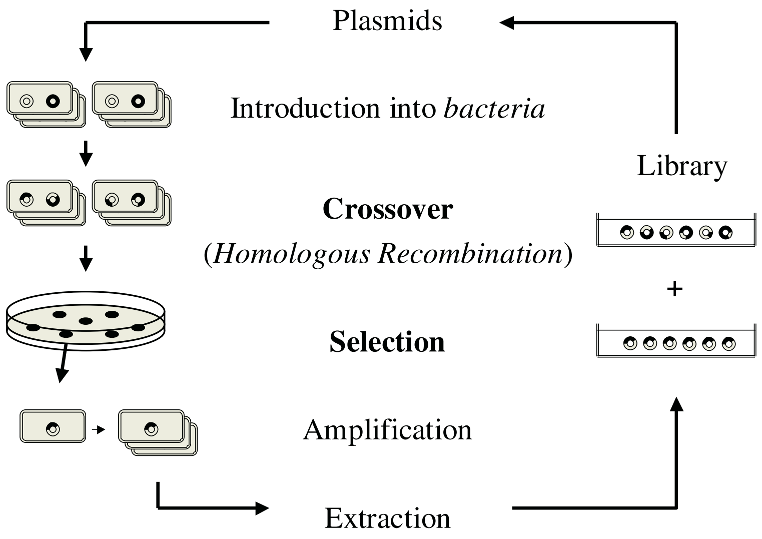


Fig. 1. Scheme Design of *CEC*

Bacterial cell colonies are picked up according to their fitness (selection). The plasmids are then amplified, extracted and returned to the library. An optimization of gene network is accomplished by applying this procedure over many generations.

2.1 Plasmid DNA Memory for CEC

Coding. A pair of plasmids, *male* and *female*, is constructed (Figure 2(a)). Sexuality does not indicate a class of F plasmid, it is merely theoretically assumed in this section. The *male* and *female* plasmids are equivalent and interchangeable except for the region of antibiotic resistance gene. A plasmid memory is composed of an array of genes. All plasmids have same set of genes and only these promoter regions are variable. Various promoter sequences can be ligated into the promoter regions of the genes by substituting its native sequences. A protein-encoding region is kept constant in order that homologous recombination reaction occurs between identical sequences of different plasmids in the host cell.

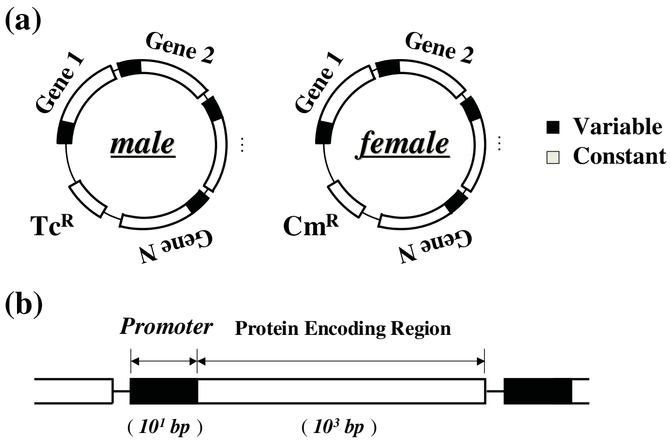


Fig. 2. Configuration of Plasmid DNA Memory

2.2 Initial Population

At the beginning of computation, CEC needs at least several number of plasmids. It would be enough to construct plasmids as many number as the variety of given promoter sequences. It is preferable to even out the proportion of *male* and *female*.

2.3 Crossover

Phase 1. The plasmid memories are introduced to the *E. coli* by the calcium chloride methods. Cells are cultured under presence of both antibiotic substances (Tc/Cm). All cells except for ones having both *male* and *female* plasmids are killed by the antibiotic substances. Crossover between *male* and *female* plasmid is carried out by the host cells in this period. One point crossover between circular plasmids results in generating a double length fusion plasmid. After overnight incubation, all plasmids are extracted from cells and recovered into a test tube.

Phase 2. Plasmids are introduced to the newly prepared *E. coli*. Cells are incubated under presence of both antibiotic substances (Tc/Cm). The fusion plasmids would be dominant at this point. In this period, another one point crossover is carried out on the fusion plasmids at a certain frequency. This second crossover occurs between the homologous sequences of male and female on the same fusion plasmid that results in generating a pair of single size plasmids. After over night incubation, all plasmids are extracted and recovered into a test tube.

Phase 3. This step is designed to delete residual fusion plasmids from the solution. The extracts are split into two fractions and digested by different restriction

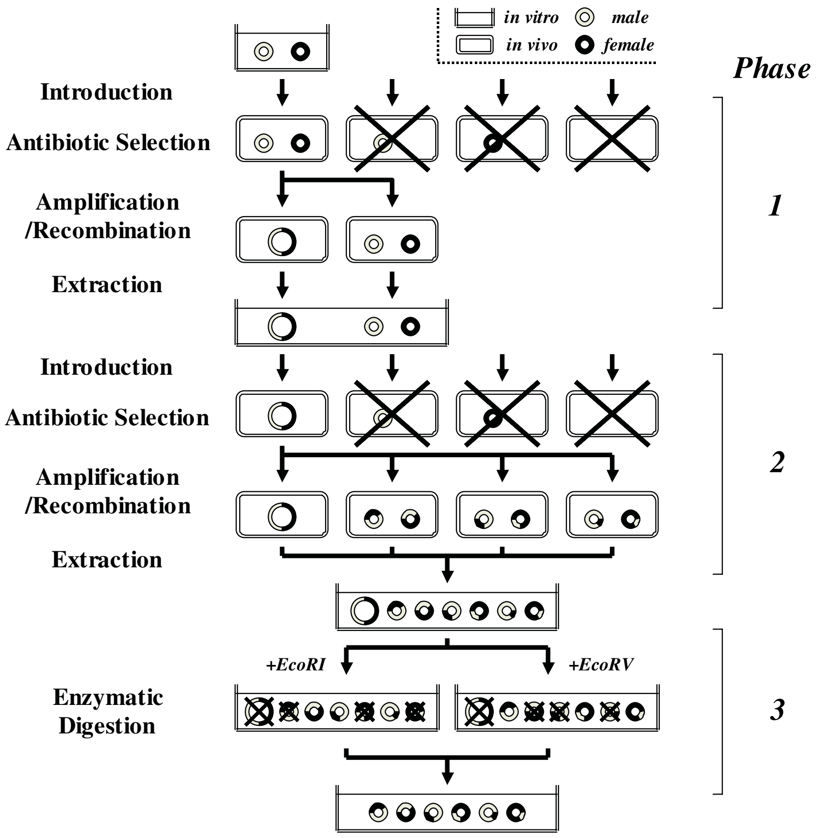


Fig. 3. Procedure of Crossover Operation

enzymes *EcoR*V and *EcoR*I, respectively. *EcoR*V cuts a restriction site on the antibiotic resistance gene Tc^R. *EcoR*I cuts a site on Cm^R gene. Double-size fusion plasmid could be digested in both case. After deactivation of the enzymes, two fractions are combined. Linearized plasmids by these enzymes are dissolved by the nuclease activity of *E. coli* in the next step.

2.4 Selection

The plasmids are introduced to the cell again. The transformed cells are dispersed to the LB agarose plate and incubated overnight. Then, the cells form colonies on the plate. All colonies are rated according to an objective fitness function. The criterion for evaluation on CEC is something like an efficiency of cellular function, a reaction activity of enzymes, a cell viability, a proliferation rate, and the like. The evaluation methods vary according to individual cases. The colonies making high scores are picked up and multiplied in the LB fluid culture.

2.5 Library Operation

The Plasmid Library keeps plasmids of older generations with wide distribution. To avoid an early convergence, the selected plasmids are combined to a plasmid library at the final step of each generation.

3 Preliminary Experiment

3.1 Procedures

Plasmid Construction. To confirm a feasibility of the crossover model, we created a pair of plasmids with two bits of memories (Figure 4a). pBR-*x1y0* and pBR-*x0y1* indicating [1, 0] and [0, 1] as the initial state were constructed from a plasmid pBR328. In order to simplify the detection process, the memory states were associated with sensitivity to antibiotic substances and different restriction sites (Table 1). pBR-*x0y1* is resistant to chloramphenicol (Cm^R) and sensitive to tetracycline (Tc^S). pBR-*x1y0* is sensitive to Cm (Cm^S) and resistant to Tc (Tc^R). Each recognition sequence of restriction endonuclease *Bgl*II and *Bln*I was integrated in the region *x1* and *y1*, respectively. *x0* and *y0*, a part of native sequence Cm^R and Tc^R, contains restriction site of *EcoR*I and *EcoR*V.

Table 1. Memory Sequences and associated phenotypes

Sequence	Length (bp)	Restriction Site	Antibiotic Resistance
x0	509	*EcoR*I	Cm^R
x1	207	*Bgl*II	Cm^S
y0	499	*EcoR*V	Tc^R
y1	153	*Bln*I	Tc^S

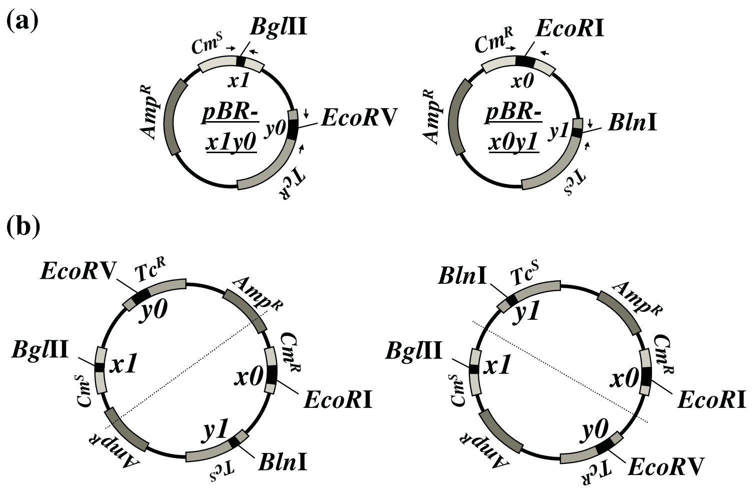


Fig. 4. (a) Plasmids for preliminary experiment. Amp^R, Cm^R and Tc^R indicate antibiotic resistance genes. Cm^S and Tc^S indicate nonfunctional genes produced by replasing *x0* and *y0* to *x1* and *y1*, respectively. Arrows indicate primer annealing sites of PCR. (b) Predicted structures of fusion plasmid. Two possible structures are shown

Crossover. The crossover experiment was performed according to the model described in the previous section. *E. coli* strain DH5α was used. We also tested ES1301(*recA*$^+$) and observed a similar result.

Detection. In order to detect the recombination, the extracted plasmids were re-introduced to the newly prepared *E. coli* DH5α after treatment of restriction enzymes. The participating enzymes are summarized in Table 2. Bacterial cells were dispersed to the LB agarose plate and incubated overnight. Some of the colonies on the plate were picked up and respectively cultured in fluid LB. After overnight culture, the plasmids were extracted from each culture. The DNA sequences of memory regions were amplified by polymerase chain reaction and electrophoresed in an agarose gel.

Table 2. Detection patterns by restriction enzymes

Restriction Enzyme	Detection Pattern [x y]
None (control)	**[00]**, **[01]**, **[10]**, **[11]**, **[10-01]**
+*Bgl*II, *Bln*I	**[00]**
+*Bgl*II, *Eco*RV	**[01]**
+*Eco*RI, *Bln*I	**[10]**
+*Eco*RI, *Eco*RV	**[11]**
+*Bgl*II, *Eco*RI *Bln*I, *Eco*RV	-

3.2 Results of Preliminary Experiment

The result of crossover experiment was shown in Figure 5. The frequency of appearance of each child was relatively equivalent although it contained some degree of bias (Figure 6). This bias was probably resulted from our mistake of design in that we associated the memories with the functional genes that had strong correlation with cell viability. The fact indicates the need to introduce an additional repressor system to control gene expression. The plasmid should be designed that its gene expression is repressed during the crossover process to avoid a bias behavior. Meanwhile it should be recovered during the evaluation and selection process. Or, simply we should choice mildest genes for optimization.

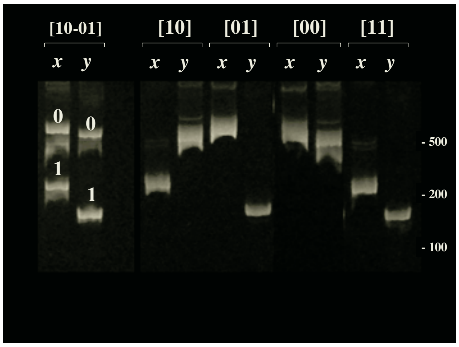


Fig. 5. Memory states of a fusion plasmid and its crossover products. Size of each PCR product is summarized in Table 1. [10-01] indicates a fusion plasmid of pBR-*x1y0* and pBR-*x0y1*

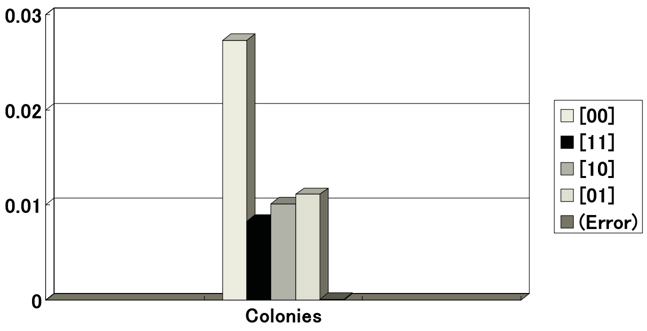


Fig. 6. Relative frequency of appearance of each crossover product. Frequency was estimated by calculating the ratio of each colony number to a number of control

4 Discussion

4.1 Crossover

This two-phased crossover method is designed to minimize the loss of population variety. Considering that a homologous recombination is supposed to process a two-point crossover, it appears that we can get single-size recombinant plasmids directly from the *Phase 1* extract. However, full reaction occurs with relatively low frequency. The extract from *Phase 1* includes relatively narrow variety of single-size recombinants. On the other hand, more number of double-size plasmids are generated by one-point crossover via defective homologous recombination. It is desirable to delete the double-size fusion plasmids from the solution at the final stage because they are out of standard in format and something illegal for optimization purposes. However, they are allowed to grow at first to maintain the diversification of the population. After generating single-size plasmids from the fusion plasmids, they are then deleted as described in *phase 3*. Another reason for allowing fusion plasmids is that they are distinguishable from single-size plasmids. Because of low frequency of crossover rate (around 3-5%), most of parent plasmids are left unchanged. We need to remove them from the final solution to achieve an equal distribution to the next generation. However, we cannot distinguish parents and children directly. By forming fusion plasmids, we can distinguish them from their children and come to be able to remove them from the final solution that leads an efficient generation shift.

4.2 Initial Population

In GA, it is required to generate randomized initial population at the beginning of the computation. However, it is not always easy to follow it on molecular-based computing. Assembling highly randomized library is one of the underlying issues of molecular-based computing. It is known that the most favorite methods such as sexual PCR and ligation-based methods tend to generate biased diversification into the library. In addition, considering the fact that the plasmid that integrates multiple genes readily reach a length of more than 10kbp, different technology other than those *in vitro* methods is required. Since the homologous recombination system processed by the cell is applicable to even longer DNAs and it appears to provide relatively less biased library, we adopt a crossover-based strategy to generate an initial population before starting CEC. Several variety of plasmids are prepared at first. This pre-initial population is then diversified by the crossover. The generated library would be used as an actual initial population of the computation.

4.3 Population Size

Unlike GA *in sillico*, a population size of CEC is not constant. All individuals need to be amplified at each step of CEC in order to resist the experimental loss. A loss of population arises in the step of introducing plasmids to *E. coli* that becomes a major bottleneck of population size. Another loss occurs in the *in vitro* step of enzymatic digestion. Altogether, around 10^3 - 10^5 would be a bottleneck size in crossover process. The reduction of population size is more serious in a selection process. The step of picking up colonies in the most typical method is carried out by hand. Hence the population size in this step is restricted to 10^1 at most. It seems that some more breakthroughs are required to remove this bottleneck.

4.4 Size of Max Generation

Although a fully autonomous computing system would be ideal, CEC on present form includes many manual works. That restricts its generation size to around 10^1. It is enough if we are to optimize small number of genes, but not sufficient to maxmize full computational power of CEC, It needs some devices to automate the system in the future.

4.5 Effect of Mutation and Other Points

Random mutation seems to influence negative effect on the crossover system of CEC. The natural process of accumulation of point mutations would be expected to reduce homology between plasmids in an evolving population that might degrade the crossover frequency. Moreover, mutation would be expected to affect the process of enzymatic digestion. Enzymatic digestion by *EcoR*V or *EcoR*I will not occur for the subset of plasmids which have undergone protecting at point mutation at the target sequence. The simplest solution to address it is integrating the multiple copies of the target sequences in the plasmid. To add to this, recovering the mutation by semiconservative crossover between selected plasmids and library plasmids, excluding homology-reduced, less evolved plasmids under a selection pressure of stepwise evolution, and maintaining the population diversity would be a key to answer these problems.

Although it is unclear if and how frequently the recombination between non-homologous region may or may not occur (e.g. between region of Tc^R and Cm^R or between sequences on plasmid and genomic DNA), we estimate its frequency will be small if any as compared to that between homologous sequences.

A low crossover frequency would scale down the bottleneck size of population around 10^{-2}. The frequency might be partially improved by inducing upregulation of *recA* by UV irradiation [12].

5 Conclusion

Genes are often influenced by many other genes in the cellular environment. Therefore, gene functions should be optimized under presence of all other genes including many seemingly unrelated genes in the cell. It is not easy to figure out all of these underlying interactions in computer simulation. In such cases, we have to optimize genes in the cell via evolutionary computational approach with real molecules and cells. The approach would become more important where the target genes are deeply involved in the cellular gene network - (e.g., the genes of multi-functional proteins, toxic proteins, DNA-binding proteins, etc.).

References

1. Bäck T, Kok JN., and Rozenberg G., Evolutionary computation as a paradigm for DNA-based computing. *Preliminary Proceedings DIMACS Workshop on Evolution as Computation*, 67–88 (1999).
2. Wood, DH., Chen, J., Antipov, E., Lemieux, B. and Cedeno, W., *In vitro* selection for a OneMax DNA evolutionary computation. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 54, 23–37 (2000).
3. Rose, JA., Takano, M., Hagiya, M., and Suyama, A., A DNA computing-based genetic program for *in vitro* protein evolution via constrained pseudomodule shuffling, *Journal of Genetic Programming and Evolvable Machines*, 4, 139–152 (2003).
4. Stemmer, W., Rapid evolution of a protein *in vitro* by DNA shuffling, *Nature* 370, 389–391 (1994).
5. Kitamura, K., Kinoshita, Y., Narasaki, S., Nemoto, N., Hushimi, Y., and Nishigaki, K., Construction of block-shuffled libraries of DNA for evolutionary protein engineering: Y-ligation-based block shuffling, *Protein Engineering*, 15 (10), 843–853 (2002).
6. Smith, GP., Filamentous fusion phage: novel expression vectors that display cloned antigens on the virion surface, *Science*, 228, 1315–1317 (1985).
7. Mattheakis, LC., An *in vitro* polysome display system for identifying ligands from very large peptide libraries, *Proc. Natl. Acad. Sci. USA*, 91, 9022–9026 (1994).
8. Nemoto, N., Miyamoto-Sato, E., Husimi, Y., and Yanagawa, H., *In vitro* virus: Bonding of mRNA bearing puromycin at the 3f-terminal end to the C-terminal end of its encoded protein on the ribosome *in vitro*, *FEBS Lett*, 414, 405–408 (1997).
9. Sakamoto K., Yamamura M., and Someya H., Toward "wet" implementation of genetic algorithm for protein engineering, *Proceedings of 10th international meeting on DNA computing*, 416–425 (2004).
10. Yokobayashi Y., Weiss R., and Arnold FH., Directed evolution of a genetic circuit, *Proc. Natl. Acad. Sci. USA* 99, 16587–16591 (2002).
11. Kitano, H., Computational Systems Biology, *Nature* 420, 206–210 (2002).
12. Salles, B. and Paoletti, C., Control of UV induction of recA protein, *Proc. Natl. Acad. Sci. USA* 80, 65–69 (1983).

An Inexpensive LED-Based Fluorometer Used to Study a Hairpin-Based DNA Nanomachine

Hanwen Yan

Staples High School, 70 North Avenue,
Westport, CT 06880 USA
hanwen@dna.caltech.edu

Abstract. Fluorometers have become indispensable tools in the study of DNA-based nanomachines. The cost of such an instrument is usually outside the budget of a high school science department or that of an amateur scientist. This paper presents a low-cost fluorometer that can be assembled for the cost of approximately a hundred dollars. By monitoring Fluorescence Resonance Energy Transfer (FRET) from the donor dye TET to the acceptor dye TAMRA, this fluorometer has been successfully used to follow the repeated opening and closing of a DNA hairpin nanomachine. This instrument makes possible the investigation of DNA-based nanotechnology or the performance of FRET-based molecular biology experiments within a high school setting.

1 Introduction

A number of DNA-based nanomachines that can be repeatedly cycled between two or more states have been devised[1-9]. For a review see Niemeyer[10]. With the exception of Yan[5], all of these investigations used Förster (or Fluorescence) Resonance Energy Transfer (FRET) to follow the conformational changes the nanodevices undergo during the course of operation. In a recent review, Jares-Erijman and Jovin[11] classified FRET techniques into 22 different methods. All of these methods take advantage of the fact that a fluorescent dye molecule (called the donor) excited by some light source can transfer its energy to a second dye molecule (called the acceptor), provided the second dye molecule has an absorption band overlapping with the emission band of the first dye. This results in a lower fluorescence from the excited dye molecule and usually an increased emission of light from the acceptor dye. The rate of energy transfer between the two molecules is a strong function of the distance between them. Thus, detecting either the change in fluorescence of the donor dye or the change in fluorescence of the acceptor dye or both allows one to infer configuration changes in nanostructures to which the dyes are attached.

The fluorometer described here allows for FRET measurements in which only the fluorescence of the donor dye is monitored because of the specific filters. Data are presented which show that this fluorometer is able to follow the opening and closing of hairpin-based DNA nanostructures. Spectrophotometers and laser-based fluorometers used for FRET studies are often in the $20,000 to $100,000 dollar price range, putting such instruments out of the range of a typical high school budget or a hobbyist's budget. The fluorometer described here can be built for a roughly one hundred dollars. It thus opens up the possibility of carrying out the growing fields of DNA-based nanotechnology investigations or FRET-based molecular biology investigation in a high school or amateur scientist setting.

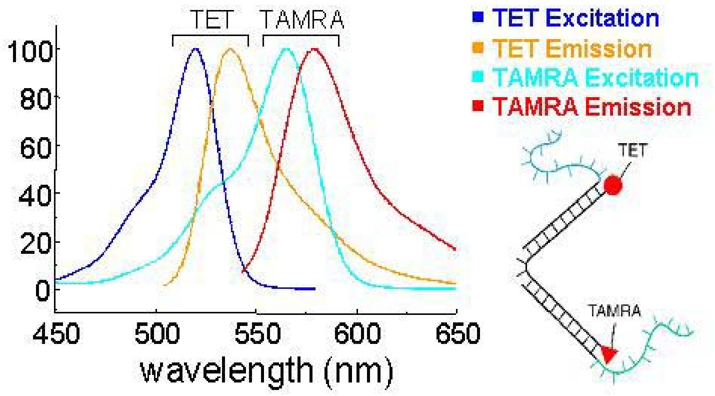


Fig. 1. Excitation and emission spectrums for fluorescent dyes, TET and TAMRA

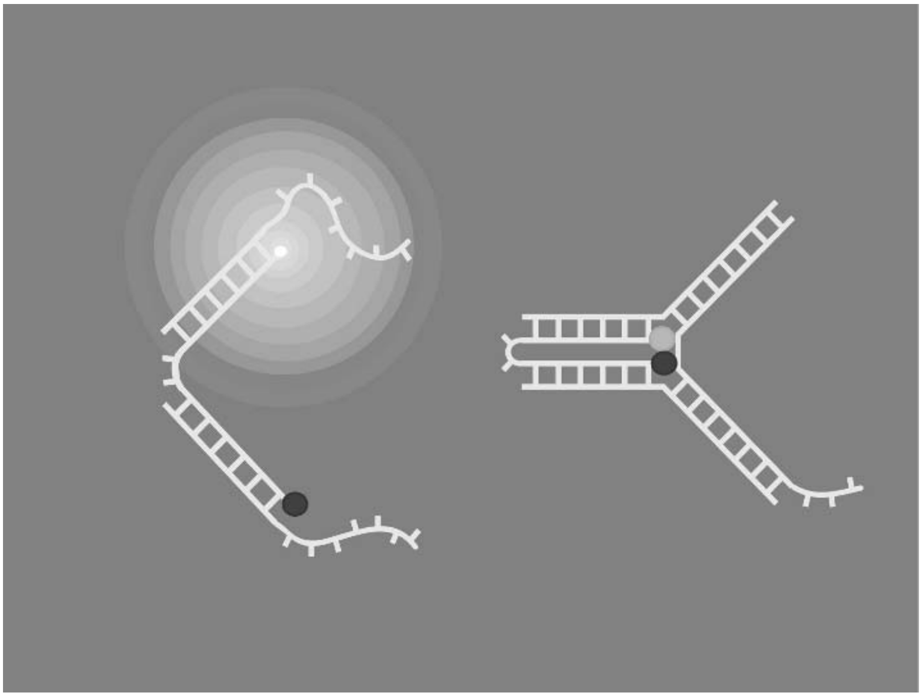


Fig. 2. DNA tweezers[2]. The excited fluorescent dye emits fluorescence when it is separated from the accepter on the left side. Resonance energy transfer from the donor to the accepter when the dyes are closer reduces the fluorescence on the right side

2 Materials and Methods – Laser-Based System

In DNA nanotechnology, FRET is used to measure the relative fluorescence of the donor dye attached to DNA oligomers. Figure 1 shows overlapping spectra for two commonly used dyes, Tetrachloro-Fluorescein (TET) and Tetramethyl Rhodamine (TAMRA).

Using a fluorometer, one may measure the fluorescence transfer from one dye, the "donor" to its counterpart, the "accepter", which quenches the donor's fluorescent emission as the distance between the two is reduced. Varying fluorescence emissions may show the change in displacement between the dyes, representing molecular motion of the DNA. Measuring fluorescence emitted by dyes attached to the DNA strands effectively represents the distance between the dyes. Figure 2 shows the fluorescence of the donor as a function of the distance between the molecules.

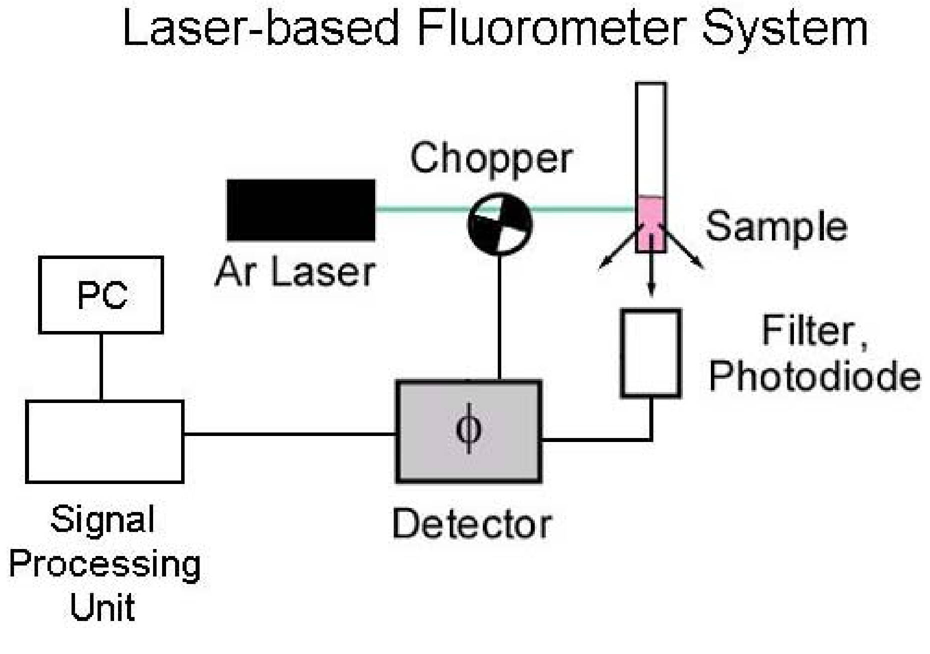


Fig. 3. A laser-based fluorometer system. In the above system, an Argon laser device projects a 514 nm wavelength – near TET's excitation spectrum – beam of light, which is sent to a chopper that splits the beam at an adjustable frequency. The resulting pulsed light is to avoid interference from other light sources, such as sunlight and 60 Hz light. The laser light is chosen to match the excitation wavelength of TET. The light then excites the TET in the DNA sample. The optical band-pass filter optimizes the fluorescence emitted by the sample, which is then detected by the photodiode. The laser and optical filter in the system are spectrum-specific for certain dyes; different dyes would require a different laser and filter. The signal processing system combines the signals from the light emission system and the photodetector system. The output of the signal processing system is linked to a PC running programs allowing analysis of the data through graphs of the FRET versus time. This system was used in DNA experiments concerning FRET and is the basis of the LED-based fluorometer design

For a laser-based fluorometer similar to the one employed by Yurke's group2-4, 7, there are three main parts: the light emission system (the laser and chopper), the photodetector system (filters and photodiodes), and the electronic signal processing system. The setup is shown in Figure 3.

3 Materials and Methods – LED-Based System

Because the expensive design of the laser-based fluorometer system is unrealistic for a high school environment, a fluorometer with a Light Emitting Diode (LED) as the light source instead of a laser would be much less expensive. This simple alteration of light source changes the entire system dramatically, especially economically, as LEDs are one of the most economical light sources available. With an LED-based system, instead of having beam splitters and optical equipment, almost the entire system becomes electronics-based which also reduces the cost. A diagram of a potential LED-based fluorometer system is shown in Figure 4.

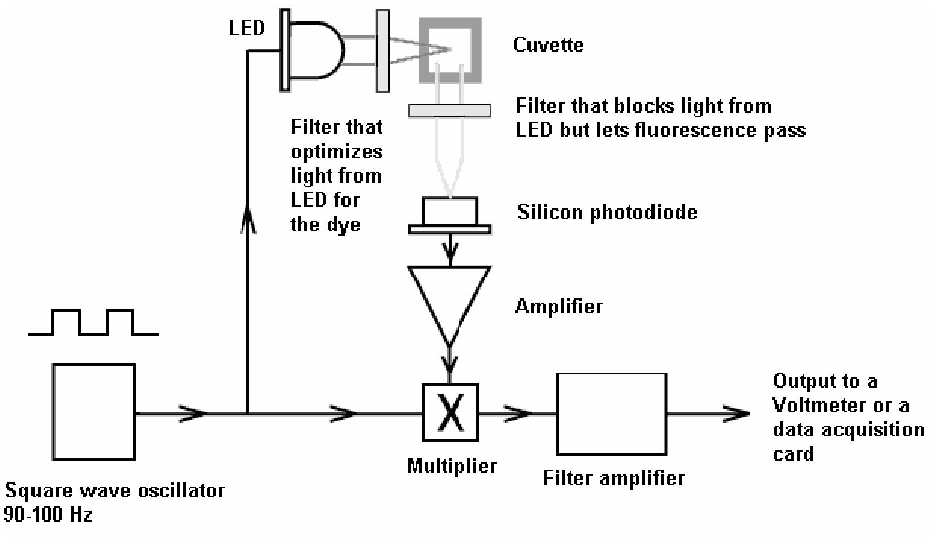

LED
Cuvette
Filter that blocks light from LED but lets fluorescence pass
Filter that optimizes light from LED for the dye
Silicon photodiode
Amplifier
Output to a Voltmeter or a data acquisition card
Multiplier
Filter amplifier
Square wave oscillator 90-100 Hz

Fig. 4. A suggested LED-based fluorometer system founded on the laser-based system. A high luminance, 520 nm LED is used to generate the light source. A square-wave oscillator drives the LED to pulse light at 90 to 100 Hz in order to avoid light interference from steady light and 60 Hz harmonics. A 525 nm band-pass filter is used for the light in order to optimally excite the TET. The fluorescence from the excited dyes is passed through a 545 nm long-pass filter which optimizes the fluorescence. A photodiode detects the fluorescence and produces an electric output signal, which is amplified through a low-noise, high-gain amplifier. This signal is further processed through the multiplier circuit, which multiplies it by the reference signal from the oscillator to generate a DC signal. This resulting signal is directly correlated to the level of the fluorescence detected. This signal is low-pass filtered and its levels are adjusted and can be displayed on a digital display or converted from analog to digital through a data acquisition system, then interfaced with a PC. Appendix B provides details of component sources

This system works in a similar manner compared to the laser-based system as shown in a comparison of the two in Table 1.

Table 1. A comparison of the fluorometer systems by functions of the system. Each system utilizes similar mechanisms to perform the same essential function

Function	Laser-based Fluorometer System	LED-based Fluorometer System
Light Source	Laser device	LED
Square Wave Generator	Chopper	Square-wave Oscillator
Light Detection	Filters and Photodiode	Filters and Photodiode
Detector System	Detector Hardware	Multiplier and Amplifier

The building of the fluorometer consisted of two phases. The first phase was to create a breadboard prototype for concept verification as shown in Figure 5. The prototype was build by assembling components onto a breadboard according to the schematics, with calculated values of resistors and capacitors (Appendix A). After the prototype was tested using an oscilloscope and multimeter, the correct values of the components could then be measured and correctly incorporated into the electronics schematics. The breadboard prototype was created to verify the concept of the LED-based system.

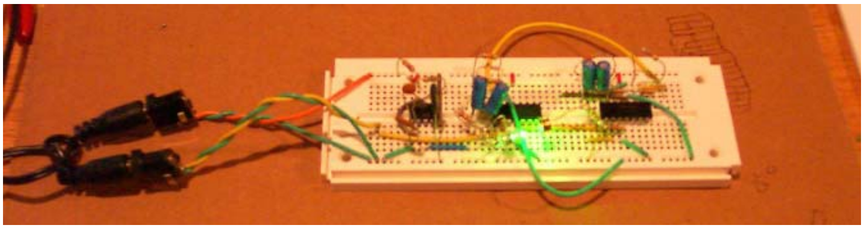


Fig. 5. A working prototype created from components placed onto a breadboard

The second phase was to build the fluorometer with a printed circuit board (PCB), which would eliminate most of the wire connections and achieve a higher signal-to-noise ratio. This is critical because of the extremely high amplifier gain. The printed circuit board was designed using ExpressPCB, a free program that allows users to create a design file and send it to the company, which fabricates the PCB for the user. The PCB was designed according to the schematics and components were then soldered onto it. A 4-digit display from Digi-Key was connected to the system to display the DC signal, reflecting the measured fluorescence. After completing the system's circuitry, a metal project box was used for mounting it, as well the sample chamber to hold the DNA sample, completing the fluorometer system. The working LED-based fluorometer system is shown in Figure 6.

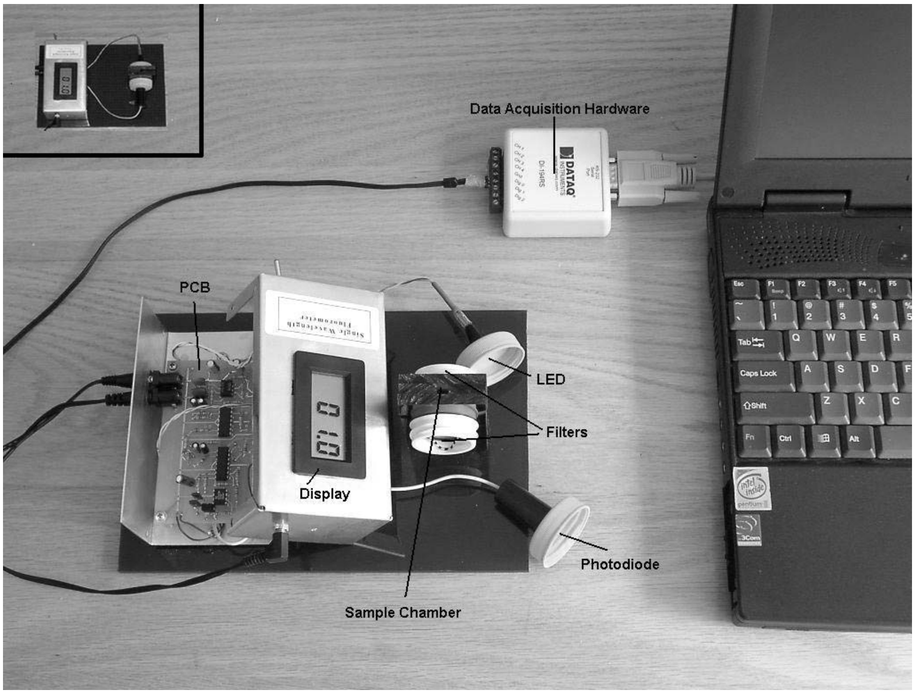


Fig. 6. The working LED-based fluorometer system, including an interface allowing connections to computer based data collection software. The assembled system is shown in the inset (top-left corner)

The LED (Appendix B) was connected to the chamber for the DNA sample opposite to the photodiode. The fluorometer is connected to the optional data acquisition system that allows connection to a PC with data acquisition software. A PC interface allows more thorough analysis of the collected data through graphing and is highly recommended, although optional.

4 Results

When working with the laser-based fluorometer, data of an unpublished DNA “hairpin” machine was obtained, an early version of DNA nanomachines that consisted of a single strand that bonded to itself to form a closed state and opened by the addition of complementary strands. Figure 7 shows the DNA hairpin nanomachine.

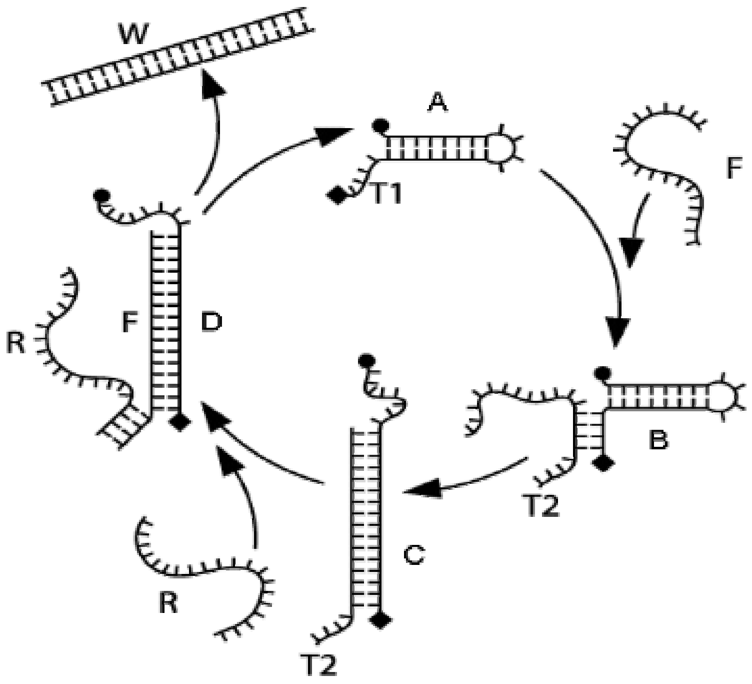


Fig. 7. The DNA hairpin nanomachine. In state A, the hairpin is bonded to itself. When the F strand is introduced, a sequence on F is complementary to T1 of the hairpin machine and bonds to it (B), eventually displacing the closed half of the hairpin and achieving the "open" state (C). When the R strand is introduced, a sequence on R is complementary to T2 of the F strand and bonds to it (D), eventually displacing the F strand from the machine and reverting to the original "closed" state (A). W is the waste product formed by F and R after cycling through the machine movement. The machine will be published in the future

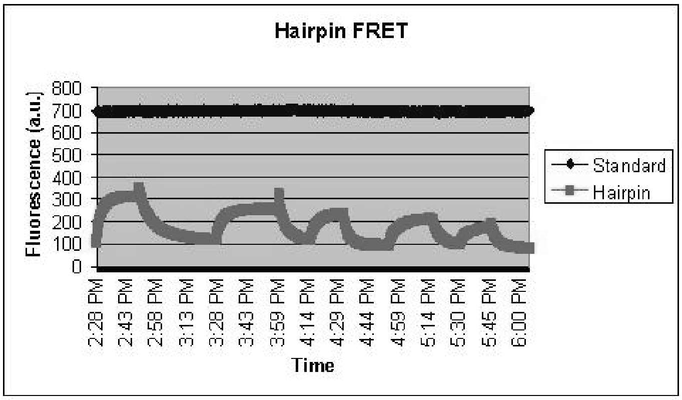


Fig. 8. Graph of DNA hairpin FRET obtained from a laser-based fluorometer system. The top line represents the standard TET used, which remained constant throughout the experiment, while the hairpin fluorescence varied

The graph in Figure 8 shows the cyclic function of the DNA hairpin. The increasing fluorescence directly correlates with the distance between the dyes, as the distance increases, so does the fluorescence. The fluorescence gradually decreases as the solution's concentration of TET decreases with the repeated addition of complement and anti-complement strands.

Figure 9 depicts two cycles of the same hairpin experiment on the LED-based fluorometer.

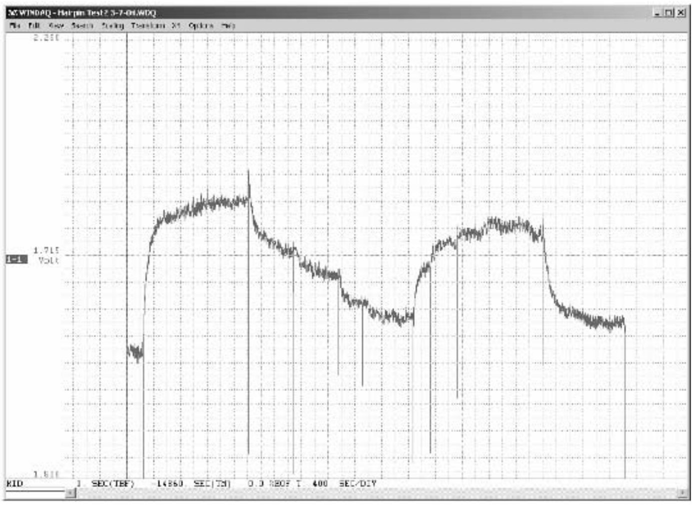


Fig. 9. The DNA hairpin FRET obtained from the LED-based system. The time interval is 400 sec/div. The fluorescence increases when F was added and decreased when its complementary R is added. The extended lines are due to the pipette inserted into the sample for mixing and adding strands

These signals demonstrate the LED-based fluorometer's ability to excite DNA conjugated fluorophores and detect the resulting emission, in this case the movement of the DNA hairpin nanomachine.

5 Discussion/Conclusion

From the graph obtained through the LED-based fluorometer, it is not only able to measure fluorescence with a high signal-to-noise ratio, but also has high detection sensitivity due to the circuit's photodiode and amplification. There are still possibilities for reduction of noise through using a better power supply. This data shows that a high-luminance LED can effectively replace a laser in a fluorometer system and still provide results similar to data obtained by a research-grade fluorometer.

The LED-based fluorometer system does not have a temperature regulation function; temperature is one variable that may affect measurements of DNA. However, results can be accurately measured with DNA at constant room temperature. Further testing with the fluorometer would involve reproducing experiments done on the laser-based fluorometer system on the LED-based system, or designing new nanomachines and testing them on both systems to compare the results. Reviewing the similarities and differences from results obtained from the two systems on the same experiment would allow a much more reliable comparison between the laser-based and LED-based fluorometer and help give evidence to support the principle of DNA nanomachine studies in the high school setting.

The approximate total materials cost of the prototype breadboard fluorometer was around eighty dollars. The printed circuit board version of the fluorometer costs around the same price with another twenty dollars for the display. The separate chamber used to hold the DNA samples so as to reduce variables that could affect measurements, such as movement, had a cost of around ten dollars. In comparison to laser-based or lamp-based fluorometers, an LED-based fluorometer system is much more economical.

With the development of a successful inexpensive fluorometer, the study of not only DNA nanotechnology, but also other fields utilizing the FRET method would be possible for much lower costs. This would greatly facilitate the study of DNA nanotechnology and FRET-based molecular biology by providing an inexpensive piece of equipment to research organizations, particularly high schools that could not normally afford such a dedicated piece of equipment to aid students in studies.

Acknowledgements

I would like to thank my high school teacher, Dr. A. J. Scheetz, for encouragement and guidance towards science research. I thank B. Yurke of Lucent Technologies for mentoring me on this project and for providing me with an opportunity to work for one month in his lab during the summer of 2002. I also thank him and Lucent for providing the cuvette and DNA used in this project and for allowing me to use a number of Yurke's PowerPoint presentation figures in this manuscript. I thank Dr. Michael Stanley at Chroma Technologies for supplying me with the filters. I also thank Ziyu Yan for assistance and guidance in the design and assembly of the electronics in this project.

References

1. Mao, C., Sun W., Shen, Z., Seeman, N.C.: A nanomechanical device based on the B-Z transition of DNA. Nature **297**, 144-146 (1999)
2. Yurke, B., Turberfield, A.J., Mills, Jr., A.P., Simmel, F.C. and Neumann, J.L.: A DNA-fuelled molecular machine made of DNA. Nature. **406**, 605–608 (2000)
3. Simmel, F.C., Yurke, B.: Using DNA to construct and power a nanoactuator. Phys. Rev. E **63** art. no. 041913 (2001)

4. Simmel, F.C., Yurke, B.: A DNA-based molecular device switchable between three distinct mechanical states. Appl. Phys. Lett. **80**, 883-885 (2002)
5. Yan, H., Zhang, X., Shen, Z., Seeman, N.C.: A robust DNA mechanical device controlled by hybridization topology. Nature **415**, 62-65 (2002)
6. Li, J.J., Tan, W.: A single DNA molecular nanomotor. Nano Lett. **2**, 315-318 (2002)
7. Turberfield, A.J., Yurke, B, Mills, Jr., A.P., Blake, M.I., Mitchel, J.C., Simmel, F.C.: Hybridization catalysis: controlled power for nanomachines. Phys. Rev. Lett. **90**, art. no. 118102, (2002)
8. Alberti, P., Mergny, J-L: DNA duplex-quadruples exchange as the basis for a nanomolecular machine. PNAS **100**, 1569-1573 (2003)
9. Feng, L., Park, S.H., Reif, J.H., Yan, H.: A two-state DNA lattice switched by DNA nanoactuator. Angew. Chem. Int. Ed. **42**, 4342-4346 (2003)
10. Niemeyer, C., Adler, M.: Nanomechanical Devices Based on DNA. Angew. Chem. Int. Ed. **41,** 3779-3783 (2002)
11. Jares-Erijman, E.A., Jovin, T.M.: FRET imaging. Nature Biotech. **21**, 1387-1395 (2003)

Appendix A – Circuit Calculations

Low-pass Filter Amplifier

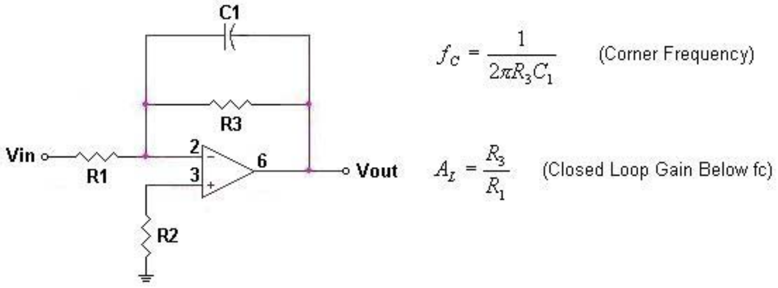


Fig. A.1. The low-pass filter amplifier used in the LED-based fluorometer system. The calculations for the values of the components are shown to the right

Multiplier Principle

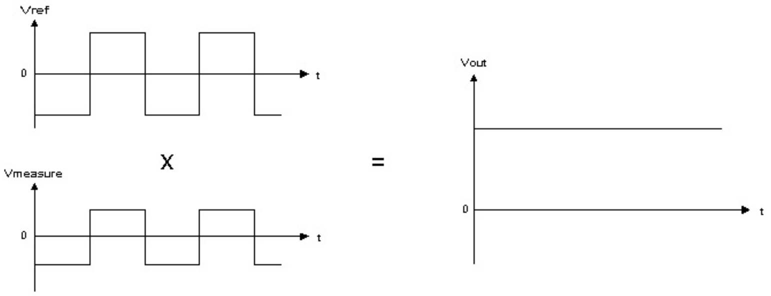


Fig. A.2. When the reference signal is multiplied by the measured signal, the result is a positive DC output signal. Both the reference signal and measured signal are independent of optical noise from the environment

Appendix B – Materials

The LED used was NSPG500S, found at The Led Light (www.theledlight.com). It was chosen for its high luminous intensity (11.6 candellas) and matching with the TET excitation spectrum (520 nm peak).

The oscillator chip used was a universal LM555 clock chip.

The Printed Circuit Board was designed using ExpressPCB, a free program from ExpressPCB (www.expresspcb.com). The company will fabricate PCBs designed with their tool.

The components (amplifier chips, capacitors, resistors, wires, etc.) were bought from Radio Shack.

The multiplier chip was Analog Devices' (www.analogdevices.com) AD633 Low Cost Analog Multiplier.

The 4-digit display was ordered from Digi-Key (www.digi-key.com). It allows the DC signal from the fluorometer system to be displayed.

The optical filters, D525/20X (band-pass) and HQ545LP (long-pass) are gifts from Chroma Technologies (www.chroma.com). The D525/20X was chosen to optimize the LED spectrum to fit the TET's excitation; it is an excitation filter with center wavelength 525 nm and is 20 nm wide, fitting TET's emission spectrum. The HQ545LP was chosen to optimize the non-fluorescent light from TET; it is a long pass filter beginning at 545 nm, near the peak of TET's emission.

The chemically synthesized DNA with fluorophores that formed the DNA hairpin was purchased from Integrated DNA Technologies (www.idtdna.com).

The data acquisition equipment was Dataq Instruments' (www.dataq.com) DI-194RS.

Designs of Autonomous Unidirectional Walking DNA Devices*

Peng Yin[1], Andrew J. Turberfield[2], and John H. Reif[1]

[1] Department of Computer Science, Duke University,
Box 90129, Durham, NC 27708-0129, USA
{py, reif}@cs.duke.edu

[2] University of Oxford, Department of Physics, Clarendon Laboratory,
Parks Road, Oxford OX 1 3PU, UK
a.turberfield@physics.ox.ac.uk

Abstract. Imagine a host of nanoscale DNA robots move autonomously over a microscale DNA nanostructure, each following a programmable route and serving as a nanoparticle and/or an information carrier. The accomplishment of this goal has many applications in nanorobotics, nano-fabrication, nano-electronics, nano-diagnostics/therapeutics, and nano-computing. Recent success in constructing large scale DNA nanostructures in a programmable way provides the structural basis to meet the above challenge. The missing link is a DNA walker that can autonomously move along a route programmably embedded in the underlying nanostructure – existing synthetic DNA mechanical devices only exhibit localized non-extensible motions such as bi-directional rotation, open/close, and contraction/extension, mediated by external environmental changes. We describe in this paper two designs of autonomous DNA walking devices in which a walker moves along a linear track unidirectionally. The track of each device consists of a periodic linear array of anchorage sites. A walker sequentially steps over the anchorages in an autonomous unidirectional way. Each walking device makes use of alternating actions of restriction enzymes and ligase to achieve unidirectional translational motion.

1 Introduction

A major challenge in nanotechnology is to precisely transport a nanoscale object from one location on a nanostructure to another location following a programmable path. DNA has been explored as an excellent building material for the construction of both large scale nanostructures and individual nanomechanical devices [11]. The successful constructions of two dimensional DNA lattices and one dimensional DNA arrays made from DX molecules [16], TX molecules [5], rhombus molecules [8], and 4x4 molecules [17] provide the structural base for realization of the above goal. However, most existing DNA nanomechanical devices only exhibit localized non-extensible motions such as open/close [7, 13, 14, 21], extension/contraction [1, 4, 6], and reversible

* Extended abstract. For full version, see [19].

rotation motion [9, 18]. Furthermore, these motions are not autonomously executed but rather mediated by external environmental changes such as the addition and removal of DNA fuel strands [1, 4, 6, 13, 14, 18, 21] or the change of ionic compositions of the solution [7, 9]. Autonomous unidirectional DNA devices executing linear translational motions are hence desirable.

There are already some exciting progress in this direction. Turberfield and colleagues have proposed to use DNA fuels to design autonomous free running DNA machines [15]. Reif has described theoretical designs of autonomous DNA walking and rolling devices that demonstrate random bidirectional translational motion along DNA tracks [10]. On the experimental side, Mao's group has recently constructed an autonomous DNA motor powered by a DNA enzyme [3]; Seeman's group has constructed a DNA walking device mediated by DNA fuel strands [12].

In the rest of the paper, we present two designs of autonomous DNA walking devices. Each device consists of a track and a walker. The track of each device contains a periodic linear array of anchorage sites. A walker sequentially steps over the anchorages in an autonomous unidirectional fashion. Each walking device makes use of alternating actions of restriction enzymes and ligase to achieve unidirectional translational motion. The action of ligase consumes ATP as energy source. The walking devices described here make the following improvements over the walking device presented in [10]. Firstly, they demonstrate unidirectional motion rather that random bidirectional motion. Secondly, the moving part (walker) in each walking device is a physical entity with a flexible body size rather than a symbolic entity, and thus the walker can serve not only as an information carrier but also as a nanoparticle carrier. These walking device designs are also different from the walking device construction by Seeman's group [12] in that they are autonomous. A limitation of our first device is that it has a low probability of falling off the track. Our second device has zero probability of falling off the track, but it is a more complicated (hence less practical) construction and assumes a restriction enzyme property that has not yet been fully-substantiated. For each walking device, we first present its structure and operation, and then describe its implementation using conceptual enzymes followed by one or more concrete examples using commercially available enzymes. The design using conceptual enzymes illustrates the general principle of the design and reveals the essential information encoding of the device that dictates its operation, while the examples using real enzymes attempt to validate the practicality of the design principles and to illustrate some technical complications in mapping the conceptual design to real enzymes.

2 Definitions

A basic structural unit used in the construction of the walking devices is a *dangler*. A dangler is a duplex DNA fragment with single strand extensions at both ends: one end is the *fixed end* that is usually attached to another structural unit (*e.g.* the backbone of the track or the body of the walker); the other end is the *sticky end*. The flexible single strand DNA at the fixed end allows the otherwise stiff dangler to move rather freely around the fixed end. This property is crucial to the operation of the devices. The fixed end only serves to structurally join a dangler to another component of the device in a flexible

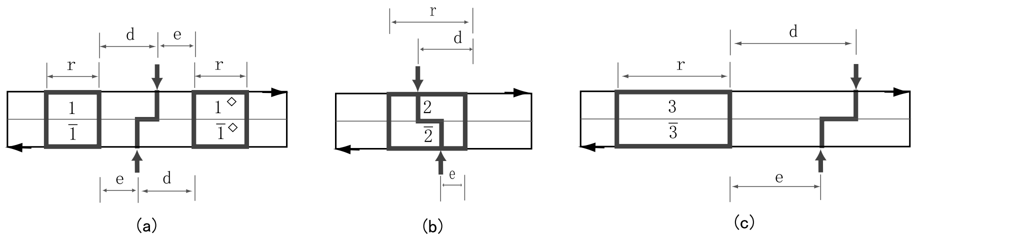


Fig. 1. Conceptual endonucleases used in the construction of the walking devices. The sequences constituting the recognition site of the endonuclease in (a), (b), and (c) are labeled with 1, $\bar{1}$, $1^\diamond$, and $\bar{1}^\diamond$; 2 and $\bar{2}$; and 3 and $\bar{3}$, respectively. Symbols r, d, and e are signed length parameters in number of bases. Recognition sites and cleavage sites are indicated with bold boxes and pairs of bold arrows, respectively. N indicates the position of a base whose value does not affect recognition by an endonuclease

fashion (*e.g.* the linkage of an anchorage to the backbone of the track/the linkage of a foot to the body a walker); the sticky end, in contrast, usually encodes information and participates actively in dictating the motion of the walker.

Two basic operational events driving the unidirectional motion of the devices are *ligations* and *cleavages*. Two neighboring danglers with complementary sticky ends can associate with each other via the hybridization of their sticky ends. Subsequent to this hybridization, the nicks at either end of the hybridized section can be sealed by a *ligase* and the two duplex fragments are hence joined covalently in a process known as *ligation*. When the context is clear, the whole process of hybridization and subsequent ligation (joining of two DNA strands) is referred to as ligation, for simplicity. In *cleavage*, an approximately reverse process to ligation, a duplex DNA fragment is cut into two separate duplex parts (with each usually possessing a complementary sticky end) by enzymes known as *restriction endonucleases*. Following cleavage, the two duplex DNA fragments (each with a sticky end) can dissociate in a process known as *melting*. When the context is clear, the whole process of cleavage and subsequent melting is referred to as cleavage. Figure 1 illustrates the conceptual restriction enzymes used in the construction of our devices. Cleavage uses no energy input from external environment while ligation consumes one molecule of ATP as energy source.

3 Device I

Design Overview. Device I consists of two parts: the *track* and the *walker*. The walker is the moving part of the device while the track is the immobile part along which the walker moves. Figure 2 (a) gives a schematic drawing of the structure of device I. The track contains a linear array of anchorages, A and B. Each anchorage is a duplex DNA fragment with a sticky end on the top, and rigidly attached to the backbone of the track. The walker stands on top of the track. The walker consists of two parts, the body and the feet (a *front foot* C and a *hind foot* D). The body is a duplex DNA segment and each foot is a DNA dangler tethered to the body via a flexible single strand DNA joint.

The flexible joint allows a foot of the walker to rove to and only to the two anchorages immediately neighboring the current anchorage on which it stands. The sticky end of a foot is complementary to the sticky end of the anchorage on which it stands and hence the foot can hybridize and be ligated with the anchorage. The ligation product between a foot and an anchorage will be cut by an endonuclease such that both the foot and the anchorage change their sticky ends. As a result, the foot will possess a sticky end that is complementary to the sticky end of the anchorage immediately ahead of the anchorage X on which the foot has been standing, but *not* complementary to the sticky end of the anchorage immediately behind X. Consequently, the foot can only hybridize and be ligated with the anchorage immediately ahead of X, but not with the one immediately behind it. This guarantees the forward motion of the walker. See Figure 2 (b) for illustration.

A foot or an anchorage X can exist in two forms, X and X^*, where $X = A$, B, C, and D. X^* is derived from X by altering its sticky end. X and X^* are required to satisfy certain properties that will be described later. At any moment during the motion, the track in front of the front foot C and behind the hind foot D consists of alternating danglers A and B^* while the track between them consists of alternating A^* and B. Assume w.l.o.g. that at the start of the motion, both feet C and D are ligated with anchorages of type A, forming A^*C and A^*D respectively. Thus the initial configuration of the walker and track complex can be written as,

$$(AB^*)_i[A^*D]B(A^*B)_j[A^*C]B^*(AB^*)_k$$

where $[A^*C]$ (resp. $[A^*D]$) is the complex between anchorage A^* and the front foot C (resp. hind foot D). To make the walker move unidirectionally down the track, we implement the following reactions between a foot and an anchorage,

$$A + C^* \xrightarrow{a} A^*C \xrightarrow{b} A^* + C \qquad B^* + C \xrightarrow{a} B^*C \xrightarrow{b} B + C^*$$

$$A^* + D \xrightarrow{a} A^*D \xrightarrow{b} A + D^* \qquad B + D^* \xrightarrow{a} BD^* \xrightarrow{b} B^* + D$$

In phase a of each reaction, a foot is ligated with an anchorage; in phase b, the foot and the anchorage are cut separate by a restriction enzyme, each now possessing a new sticky end. Applying the reactions to the walker-track complex, we have the following motion of the walker along the track,

$$(AB^*)_i[A^*D]B(A^*B)_j[A^*C]B^*(AB^*)_k$$

$$\to (AB^*)_iA[B^*D](A^*B)_jA^*[B^*C](AB^*)_k$$

$$\to (AB^*)_{i+1}[A^*D]B(A^*B)_j[A^*C]B^*(AB^*)_{k-1}$$

The above is a full induction cycle of the motion of the walker, and hence the walker can (in principle) move forward along the track infinitely. We further require that phase a of each reaction is *not* reversible, thus the whole reaction is irreversible. Consequently, the walker can only move unidirectionally along the track.

Note that the hind foot D can be viewed as the dual to the front foot C: during the motion, front foot C changes the configuration of the track from (AB^*) to A^*B; hind foot D moves on the modified track and restores it to its original configuration AB^*.

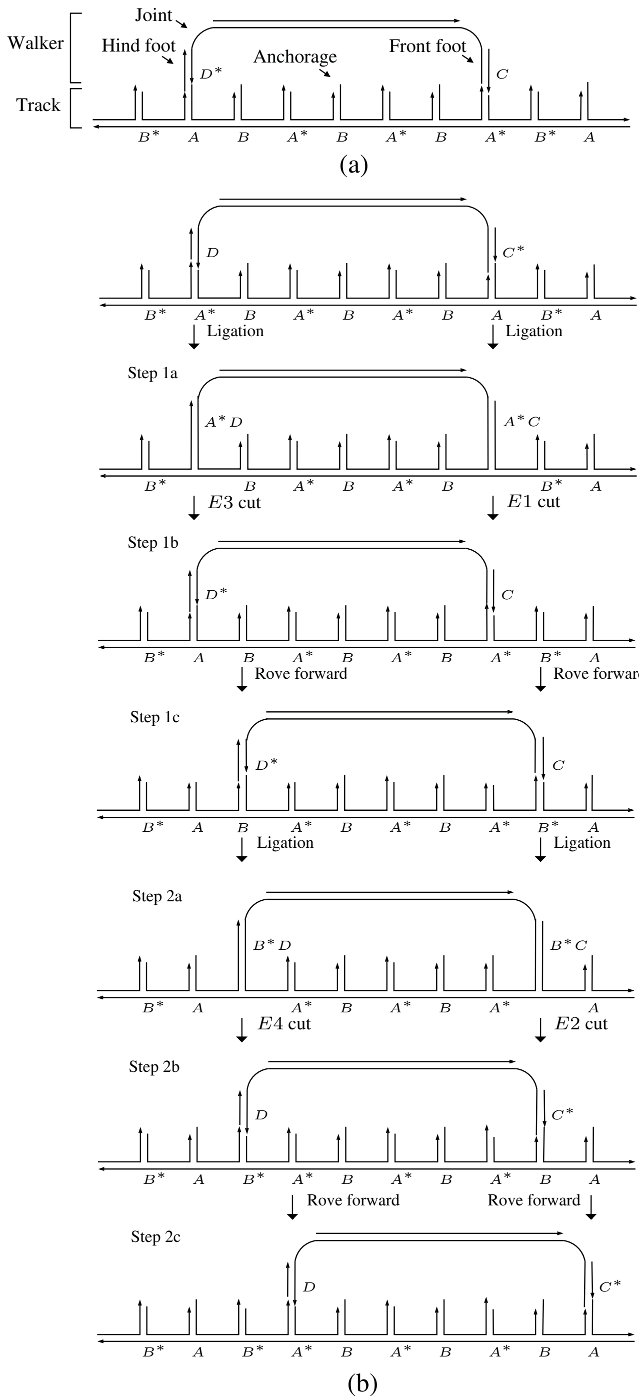


Fig. 2. The structural design and step by step operation of device I. (a) Structural design of the device. (b) Step by step operation of the device. The walker moves unidirectionally to the right

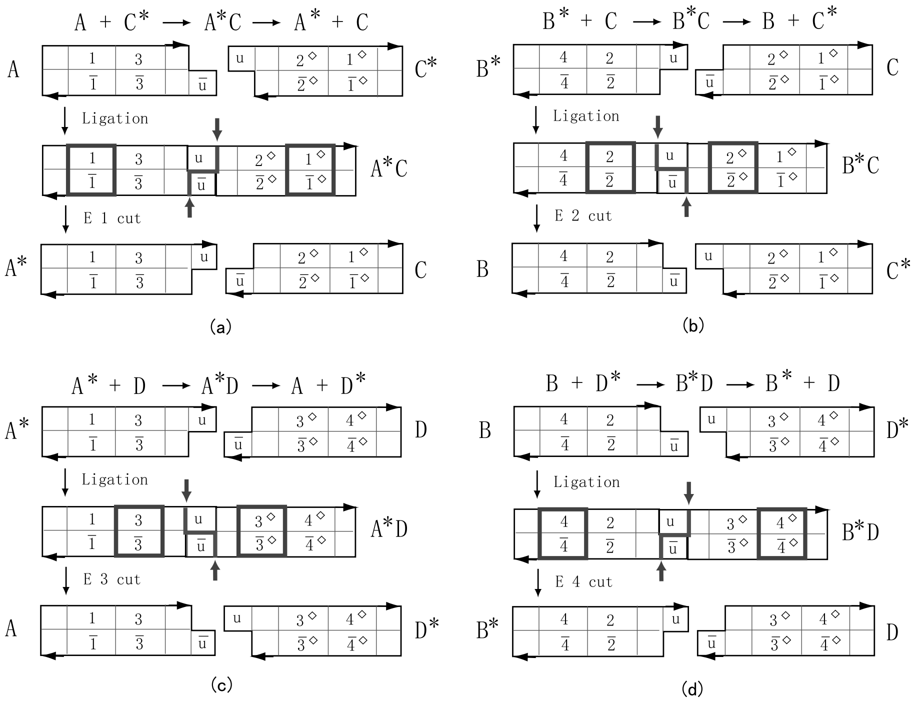


Fig. 3. Implementation of device I using four conceptual restriction enzymes. Endonuclease recognition sites and cleavage sites are indicated with bold boxes and pairs of bold arrows, respectively

Implementation with conceptual endonucleases. To implement the designed reactions, we use four conceptual enzymes $E1$, $E2$, $E3$, and $E4$. The cutting patterns of these enzymes are similar to the one depicted in Figure 1 (a). Here we require that $d_1 - e_1 = d_4 - e_4 = e_2 - d_2 = e_3 - d_3$, where d_i and e_i are the length parameters for endonuclease Ei. Figure 3 describes the detailed step by step reactions that dictate the motion of the walker. Since only the region near the end of an anchorage or a foot is relevant for the reactions, we only depict the end regions in Figure 3.

Figure 3 (a) depicts reaction $A + C^* \rightarrow A^*C \rightarrow A^* + C$. In this reaction, the sticky end $\bar{u}$ of anchorage A is first ligated with the sticky end u (complementary to $\bar{u}$) of foot C^*, generating ligation product A^*C. This corresponds to the reaction of the front foot in Step 1a in Figure 2 (b): $A + C^* \rightarrow A^*C$. A^*C contains a recognition site for endonuclease $E1$ and is cut by $E1$ into A^* and C (Step 1b in Figure 2 (b): $A^*C \rightarrow A^* + C$). Note that now front foot C possesses a new sticky end $\bar{u}$. Recall that the anchorage immediately ahead of the anchorage A^*, on which front foot C is standing, is anchorage B^*, which possesses a sticky end u (complementary to $\bar{u}$). Thus C can rove forward and hybridize with B^* (Step 1c in Figure 2 (b)). This leads to the reaction in Figure 3 (b): $B^* + C \rightarrow B^*C \rightarrow B + C^*$. First, the hybridization product between B^* and C is ligated to form B^*C (Step 2a in Figure 2 (b): $B^* + C \rightarrow B^*C$).

Table 1. Implementation of device I with endonucleases Ahd I, Fnu4H I, ScrF I, and Xcm I. Ligation sites and cleavage sites are denoted with − and ˆ, respectively. The bases that determine recognition sites in action are in upper case

Reactions	Enzymes	DNA Sequences
$A + C^* \rightarrow A^*C$	Ligase	$5'$...gaccc-ngcgtc... $3'$ $3'$...ctgggn-cgcag... $5'$
$A^*C \rightarrow A^* + C$	Ahd I	$5'$...GACcc nˆgcGTC... $3'$ $3'$...CTGggˆn cgCAG... $5'$
$B^* + C \rightarrow B^*C$	Ligase	$5'$...ccanngcn-gcgtc... $3'$ $3'$...ggtnncg-ncgcag... $5'$
$B^*C \rightarrow B + C^*$	Fnu4H I	$5'$...ccannGCˆn GCgtc... $3'$ $3'$...ggtnnCG nˆCGcag... $5'$
$A^* + D \rightarrow A^*D$	Ligase	$5'$...gacccn-ggnntgg... $3'$ $3'$...ctggg-nccnnacc... $5'$
$A^*D \rightarrow A + D^*$	ScrF I	$5'$...gacCCˆn GGnntgg... $3'$ $3'$...ctgGG nˆCCnnacc... $5'$
$B + D^* \rightarrow B^*D$	Ligase	$5'$...ccanngc-nggnntgg... $3'$ $3'$...ggtnncgn-ccnnacc... $5'$
$B^*D \rightarrow B^* + D$	Xcm I	$5'$...CCAnngc nˆggnnTGG... $3'$ $3'$...GGTnncgˆn ccnnACC... $5'$

This ligation product is subsequently cut into B and C^* by endonuclease $E2$ (Step 2b in Figure 2 (b): $B^*C \rightarrow B + C^*$). Now front foot C^* possesses sticky end u, and hence it will rove forward and hybridize with anchorage A down the track (Step 2c in Figure 2 (b)). This completes a full induction cycle for the front foot.

Note that reaction $A+C^* \rightarrow A^*C$ is irreversible: no restriction enzyme is present to cut A^*C into A and C^*. This effectively establishes the irreversibility of the motion of foot C. However, we note that after A^*C is cut into A^* and C, the two can be religated into A^*C (which is subsequently cut back into A^* and C). This represents an idling step in the motion of the walker. Similar analysis applies to the reaction $B^* + C \rightarrow B^*C \rightarrow B + C^*$.

The motion of hind foot D is similar to motion of front foot C and we omit its description for brevity.

Molecular implementation using real enzymes. We give two implementations with real enzymes. The first one is a direct mapping of the implementation using the conceptual enzymes in Figure 3. The real enzymes used are shown in Figure 4 (a). Here, real endonucleases $AhdI$, $Fnu4HI$, $ScrFI$, and $XcmI$ correspond to conceptual endonucleases $E1$, $E2$, $E3$, and $E4$, respectively. The reactions are shown in Table 1.

The second implementation reduces the number of endonucleases to three by using a non-palindromic endonuclease (Aci I) and its slightly more involved construction is shown in Table 2. The real enzymes used are shown in Figure 4 (b). Note that Aci I and Hha I correspond to the conceptual enzyme depicted in Figure 1 (b).

Processivity of device I. A key technical issue in the construction of device I is to ensure that the walker is constrained to stay on or near the track. An isolated foot C or

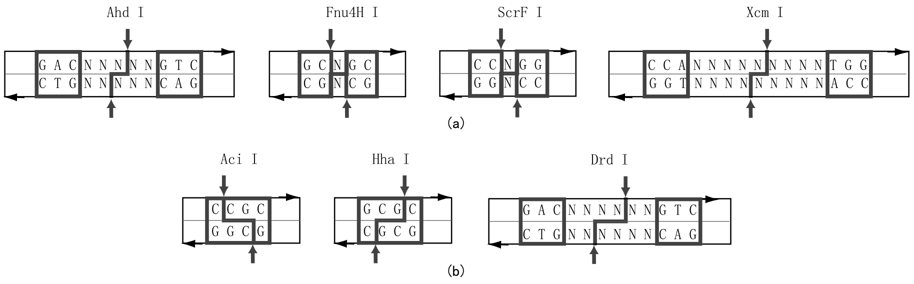


Fig. 4. Real enzymes used in the construction of device I. Endonuclease recognition sites and cleavage sites are indicated with bold boxes and pairs of bold arrows, respectively. N indicates the position of a base whose value does not affect recognition by an endonuclease

Table 2. Implementation of device I with endonucleases Aci I, Hha I, and Drd I. Ligation sites and cleavage sites are denoted with – and ˆ, respectively. The bases that determine recognition sites in action are in upper case

Reactions	Enzymes	DNA Sequences
$A + C^* \rightarrow A^*C$	Ligase	5′...gacnccg-c... 3′ 3′...ctgng-gcg... 5′
$A^*C \rightarrow A^* + C$	Aci I	5′...gacnCˆCG C... 3′ 3′...ctgnG GCˆG... 5′
$B^* + C \rightarrow B^*C$	Ligase	5′...c-cgc... 3′ 3′...cgc-g... 5′
$B^*C \rightarrow B + C^*$	Hha I	5′...G CGˆC... 3′ 3′...CˆGC G... 5′
$A^* + D \rightarrow A^*D$	Ligase	5′...gacnc-cggngtc... 3′ 3′...ctgnggc-cncag... 5′
$A^*D \rightarrow A + D^*$	Drd I	5′...GACnc cgˆgnGTC... 3′ 3′...CTGngˆgc cnCAG... 5′
$B + D^* \rightarrow B^*D$	Ligase	5′...gcg-gngtc... 3′ 3′...c-gccncag... 5′
$B^*D \rightarrow B^* + D$	Aci I	5′...GˆCG Gngtc... 3′ 3′...C GCˆCncag... 5′

D would easily fall off the track and diffuse away. However, we can reduce the falling-off probability by constructing a multi-footed walker. Instead of possessing only two feet as in Figure 2, the walker has an array of alternate C and D feet. The feet are attached to a common backbone. Hence the walker is held to the track by multiple bonds - even if none are ligated (so all bonds are weak 1- or 2-base hydrogen bonds) then the probability of detachment is small. This is precisely what is needed - feet are held in the right place with the right amount of freedom to move - it introduces the constraint that no foot can move more than two anchorages forward until all feet have moved at least one anchorage.

4 Design II

Overview. A potential problem of device I is that it may fall off the track. Though a walker with more feet has lower probability of falling off, we can not completely eliminate such risk. In contrast, the device we describe next is guaranteed to stay on the track, though it has a more complicated (hence less practical) construction and assumes a restriction enzyme property that has not yet been fully-substantiated. In device II, a two-footed walker steps over the anchorages along a track unidirectionally. The design of device II is based on the following principle: the lifting of one foot off the track is conditional on the attachment (ligation) of the other foot to the track. This attachment principle can ensure that at any moment, at least one foot of the walker is attached to the track. We describe the structure and step by step operation of device II below.

The track and the walker are depicted in Figure 5 (a). The track contains a linear array of anchorages. Each anchorage is a duplex DNA fragment with single strand DNA overhangs at both ends and its midpoint is tethered to the backbone of the track via single strand DNA. Thus the anchorage can be viewed as a two-ended dangler. In addition, between every two neighboring anchorages is tethered another dangler, referred to as a *switch*. As we shall see below, the alternating arrangement of anchorages and switches are used to construct a signaling mechanism which ensures the unidirectional and non-falling-off-track motion of the walker. The anchorages and switches are denoted as T_i and S_i respectively, where $i = 1, 2, 3, \ldots, n$. A switch S_i can only be ligated with its immediate anchorage neighbors T_{i-1} and T_i. The upper ends of T are of type C^*, and the lower end of T_i is of type A^* (B^*) for odd (even) i.

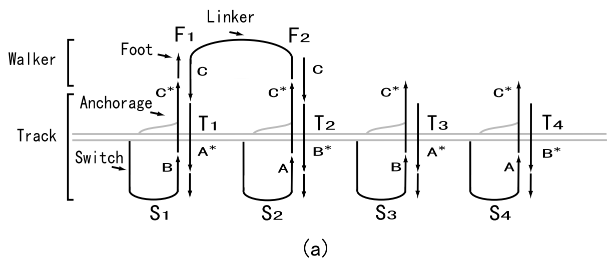
The walker consists of two danglers connected with a single strand DNA . The two danglers serve as the feet of the walker and are denoted as F_1 and F_2. The ends of both F_1 and F_2 are of type C. The walker stands on top of the upper ends of the anchorages and walks down the track unidirectionally, with the switch/anchorage complex of the road serving both as attaching points and as a signal transducing device to dictate the lifting and attaching of its feet in an alternating fashion such that it never falls off the track. In particular, at any point, if one foot is attached to anchorage T_i, the other foot can only be attached to T_i's immediate neighbors, T_{i-1} and T_{i+1}.

The ends of the feet of the walker, of the anchorages, and of the switches have the following properties:

1. The complementary end pairs are: (A, A^*), (A, B^*), (B^*, A^*), (B, B^*), and (C, C^*). Only two danglers with these complementary ends can be ligated.

2. The formation of CC^* ligation product at the upper end of the anchorage introduces a recognition site on the anchorage for endonuclease $E3$, which has a similar cleavage pattern as the one depicted in Figure 1 (c). And this results in a cleavage at the other end of the anchorage such that the anchorage is cut from the switch it is currently ligated with (if there is one). Similarly, the formation of A^*A (resp. B^*B) at the lower end of the anchorage will produce a recognition site on the anchorage for endonuclease $E1$ (resp. $E2$) and this will result in the cleavage of CC^* at the upper end of the anchorage if there is a foot end C ligated with C^*.

We will next see how these properties guarantee the desired motion of the walker as we go through a step by step description of the walker's motion.



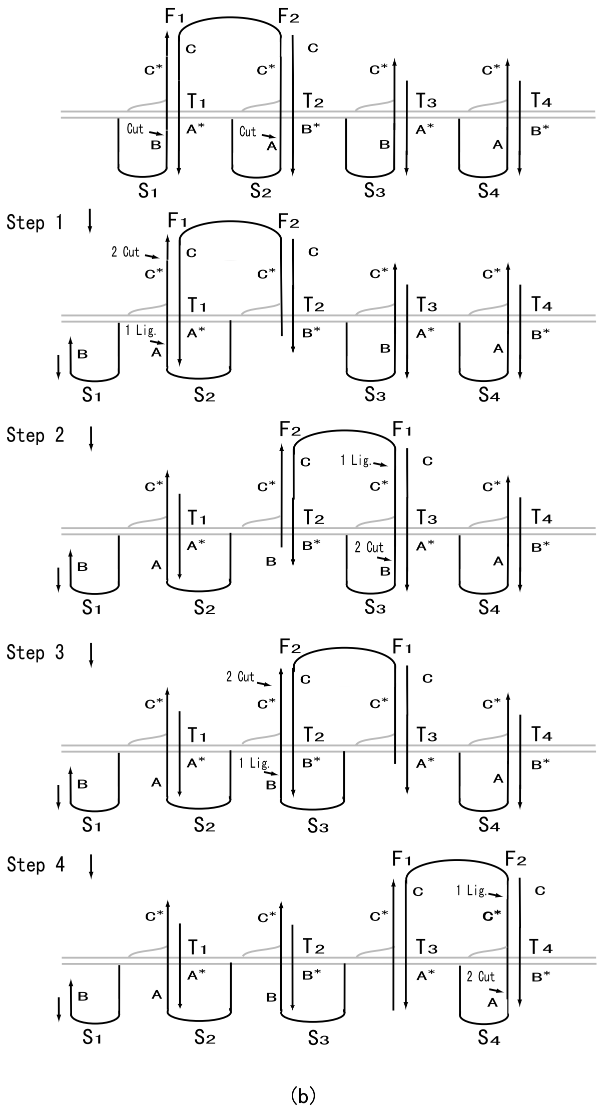


Fig. 5. The structural design and step by step operation of device II. (a) Structural design of the device. (b) Step by step operation of the device. The track is depicted in light gray

Step by step motion. Now we describe the four steps of the walker's motion that completes a full inductional cycle. Initially, the walker and track complex is assembled in such a way that the feet F_1 and F_2 of the walker are ligated with anchorages T_1 and T_2, respectively; each switch S_i is ligated to the lower end of T_i, forming BA^* for odd i and AB^* for even i. Note that BA^* and AB^* are different.

Step 0. Upon introduction of enzymes into the system, switches S_1 and S_2 are cut from anchorages T_1 and T_2 respectively, since the CC^* sequences at the upper ends of T_1 and T_2 constitute endonuclease $E3$ recognition sites and hence result in cleavages at the lower ends of T_1 and T_2. Now S_2 (with end A) can explore its neighboring space and be ligated with either T_1 (with end A^*) or T_2 (with end B^*). Ligation between S_2 and T_2 is a just an idling step: the ligation product will be subsequently cut again. In contrast, ligation of S_2 and T_1 brings the system to Step 1.

Step 1. The ligation of S_2 (with end A) and T_1 (with end A^*) introduces a recognition site for $E1$, and thus results in the cleavage of F_1 from the upper end of T_1. Note that the ligation product between the lower end of T_1 and S_2 contains recognition site (AA^*) for $E1$ while the ligation product between foot F_1 and the upper end of T_1 contains recognition site (CC^*) for endonuclease $E3$. As such, both $E1$ and $E3$ will compete to perform cleavage on the common ligation product. (See Figure 6 (a) for detail.) It is possible that endonuclease $E3$ cuts switch S_2 away from anchorage T_1, resulting in an idling step. However, there must also be non-zero probability that endonuclease $E1$ cuts foot F_1 away from anchorage T_1, advancing the system to Step 2.

Step 2. Now foot F_1 has free end C and can swing around the ligation product between foot F_2 and anchorage T_2 and get ligated with the upper end C^* of anchorage T_3. Note that now foot F_1 is in front of foot F_2. The ligation of CC^* subsequently results in the cleavage of S_3 from T_3.

Step 3. Switch S_3 has free end B and is ligated with the B^* end of anchorage T_2, and the newly formed recognition site BB^* leads to the action of endonuclease $E2$ and results in the cleavage between foot F_2 and anchorage T_2.

Step 4. Foot F_2 swings to in front of foot F_1 and is ligated with anchorage T_4, resulting in the cleavage of switch S_4 from the lower end of anchorage T_4.

Upon completion of Step 4, the walker has moved from anchorages T_1 and T_2 to anchorages T_3 and T_4. This finishes a full inductional cycle, and hence the walker can continue moving down the track.

Correctness. To show the correctness of the design, we prove the following three properties of the walker: 1) the motion of the walker is unidirectional; 2) the walker never falls off the track; 3) the motion of the walker is never blocked. We omit detail here due to lack of space, but refer the reader to [19].

Implementation with conceptual enzymes. Figure 6 describes the implementation of device II with these conceptual restriction enzymes $E1$, $E2$, and $E3$, which have similar cutting patterns as the one shown in Figure 1 (c). We require that $d_1 = d_2 = d_3$ and $e_1 = e_2 = e_3$, where d_i and e_i are the length parameters for E_i (i = 1, 2, and 3). In Figure 6 (a), two anti-parallel flows of reactions are depicted. Starting from the top, end A (of a switch) has sticky end sequence complementary to end A^* (lower end of an anchorage) and hence the two are ligated together. This creates a recognition site for

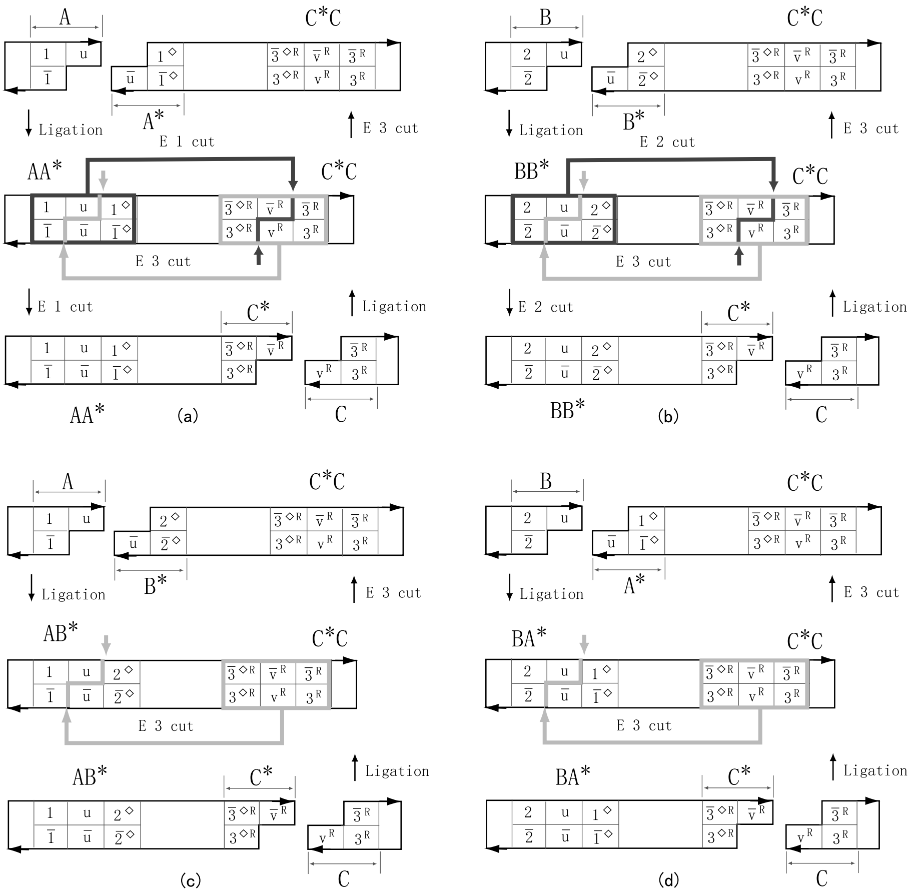


Fig. 6. Actions of conceptual enzymes used in the construction of device II. (a) Sequences 1, u, $1^\diamond$, $\bar{1}$, $\bar{u}$, and $\bar{1}^\diamond$ (sequences of AA^*) together constitute the recognition site (red box) for conceptual endonuclease $E1$, whose cleavage site is indicated with a pair of bold arrows. Sequences $\bar{3}^{\diamond R}$, $\bar{v}^R$, $\bar{3}^R$ $3^{\diamond R}$, v^R, and 3^R (sequences of C^*C) together constitute the recognition site (light gray box) for conceptual endonuclease $E3$, whose cleavage site is indicated with a pair of light gray arrows.(b) Two anti-parallel flows of reactions by $E2$ and $E3$. (c) and (d) Neither ligation of AB^* or BA^* results in cleavage of CC^*

endonuclease $E1$, and results in the cleavage of end C (of a foot) from end C^* (upper end of an anchorage). This downward flow of reactions can be fully reversed into the anti-parallel upward flow starting from the bottom with C^* and C and ends at the top with A and A^*. We note that due to the fully reversible nature of reactions, the reaction system has nonzero probability to explore all three states: the top one (A, A^* :: C^*C), the middle one (AA^* :: C^*C), and the bottom one (AA^* :: C^*, C), where :: represents the duplex portion of DNA connecting the two ends. Similar fully reversible anti-parallel flows of reactions involving $E2$ and $E3$ are depicted in Figure 6 (b). In

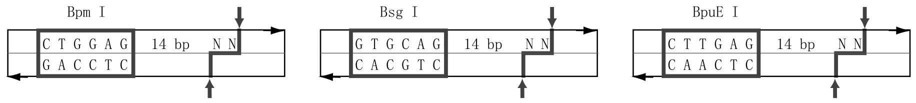


Fig. 7. Real enzymes used in the construction of device II. Endonuclease recognition sites and cleavage sites are indicated with bold boxes and pairs of bold arrows, respectively. N indicates the position of a base whose value does not affect recognition by an endonuclease

Table 3. Implementation of device II with endonucleases Bpm I, Bsg I, and BpuE I. Ligation sites and cleavage sites are denoted with – and ˆ, respectively. The bases that determine recognition sites in action are in upper case

Reactions	Enzymes	Sequences
$A + A^* :: C^*C \rightarrow$ $AA^* :: C^*C$	Ligase	$5'$...ctg-gag(n)$_{11}$ctcaag...$3'$ $3'$...g-acctc(n)$_{11}$gagttc...$3'$
$AA^* :: C^*C \rightarrow$ $AA^* :: C^* + C$	Bpm I	$5'$...CTGGAG(n)$_{11}$ctc aaˆg...$3'$ $3'$...GACCTC(n)$_{11}$gagˆtt c...$3'$
$AA^* :: C^* + C \rightarrow$ $AA^* :: C^*C$	Ligase	$5'$...ctggag(n)$_{11}$ctcaa-g...$3'$ $3'$...gacctc(n)$_{11}$gag-ttc...$3'$
$AA^* :: C^*C \rightarrow$ $A + A^* :: C^*C$	BpuE I	$5'$...c tgˆgag(n)$_{11}$CTCAAG...$3'$ $3'$...gˆac ctc(n)$_{11}$GAGTTC...$3'$
$B + B^* :: C^*C \rightarrow$ $BB^* :: C^*C$	Ligase	$5'$...gtg-cag(n)$_{11}$ctcaag...$3'$ $3'$...c-acgtc(n)$_{11}$gagttc...$3'$
$BB^* :: C^*C \rightarrow$ $BB^* :: C^* + C$	Bsg I	$5'$...GTGCAG(n)$_{11}$ctc aaˆg...$3'$ $3'$...CACGTC(n)$_{11}$gagˆtt c...$3'$
$BB^* :: C^* + C \rightarrow$ $BB^* :: C^*C$	Ligase	$5'$...gtgcag(n)$_{11}$ctcaa-g...$3'$ $3'$...cacgtc(n)$_{11}$gag-ttc...$3'$
$BB^* :: C^*C \rightarrow$ $B + B^* :: C^*C$	BpuE I	$5'$...g tgˆcag(n)$_{11}$CTCAAG...$3'$ $3'$...cˆac gtc(n)$_{11}$GAGTTC...$3'$
$AB^* :: C^* + C \rightarrow$ $AB^* :: C^*C$	Ligase	$5'$...ctgcag(n)$_{11}$ctcaa-g...$3'$ $3'$...gacgtc(n)$_{11}$gag-ttc...$3'$
$AB^* :: C^*C \rightarrow$ $A + B^* :: C^*C$	BpuE I	$5'$...c tgˆcag(n)$_{11}$CTCAAG...$3'$ $3'$...gˆac gtc(n)$_{11}$GAGTTC...$3'$
$BA^* :: C^* + C \rightarrow$ $BA^* :: C^*C$	Ligase	$5'$...gtggag(n)$_{11}$ctcaa-g...$3'$ $3'$...cacctc(n)$_{11}$gag-ttc...$3'$
$BA^* :: C^*C \rightarrow$ $B + A^* :: C^*C$	BpuE I	$5'$...g tgˆgag(n)$_{11}$CTCAAG...$3'$ $3'$...cˆac ctc(n)$_{11}$GAGTTC...$3'$

contrast, reactions in Figure 6 (c) and 6 (d) are not fully reversible since neither ligation of AB^* nor that of BA^* can result in a recognition site for an endonuclease, and hence CC^* can not be cleaved. This irreversibility ultimately accounts for the unidirectionality of the motion of the walker. The downward reaction flow in Figure 6 (a), the upward reaction flow in (d), the downward reaction flow in (b), and the upward reaction flow in (c) correspond to Steps 1, 2, 3, and 4 in Figure 5, respectively.

Molecular implementation with real enzymes. The above conceptual enzymes can be mapped directly to real enzymes in Figure 7, where conceptual enzymes $E1$, $E2$, and $E3$ correspond to real enzymes Bpm I, Bsg I, and BpuE I, respectively. Table 3 describes the implementation with these real enzymes. Note that we have the following mapping from sequences in Figure 6 to the sequences in Table 3: $1 =$ C, $u =$ TG, $1^\diamond =$ GAG, $2 =$ G, $2^\diamond =$ CAG, $\bar{3}^{\diamond R} =$ CTC, $\bar{v}^R =$ AA, and $\bar{3}^R =$ G.

Practicality. One assumption we make about the enzyme is that the presence of a single strand between the recognition site and cleavage site of each endonuclease used above will neither alter the specificity nor totally inhibit the activity of that endonuclease. A theoretical modeling of the molecular structure of the enzyme and its interaction with the DNA strands would shed light on the practicality of this assumption. However, the final validation of this assumption relies on a rigorous experimental study. Though our preliminary experimental result is in agreement with this assumption, more work is still required to further substantiate this assumption.

5 Discussion

We have depicted the backbones of the walking devices as duplex DNA fragments for simplicity. However, this is not technically precise. One property we require of the backbone of a track is its rigidity, to ensure that the walker cannot skip anchorage(s) and "jump" ahead. Existing DNA lattices provide such a platform [5, 8, 16, 17]. We can easily embed the anchorages to a rigid DNA lattice and thus integrate a walking device to a lattice, with the latter provide the desired rigid backbone for the anchorages. In addition to the rigidity of the track, the structure and the size of the walker are also crucial factors to ensure that a foot can only explore the immediately neighboring anchorages. In device I, though it is hard to ensure this property for a two-footed walker (since in such a walker one foot might swing around the other foot in a similar fashion as in device II), this property can be rather straightforwardly guaranteed in a multi-footed walker with a rigid body. In device two, the two feet of the walker alternate their order along the track by swinging around each other and we hence only need to properly design the size of the body such that a foot can only reach a neighboring anchorage.

The designs of the devices assume that enzyme cleavage occurs only after the DNA strands are ligated. This is assumption is in agreement with the experimental results observed in our recent construction of a unidirectional autonomous DNA walker [20]. In this device, we use two class II enzymes PflM I and BstAP I and the system operates at 37 $^\circ C$. However, we note that this property does not hold true for all class II enzymes under all conditions. Indeed, Shapiro's group has observed that a class II enzyme Fok I can cleave GC rich DNA duplex strands with nicks present between Fok I recognition site and cleavage site under at low temperature (8 $^\circ C$) [2].

Acknowledgement

The authors would like to thank Hao Yan and Thomas H. LaBean for helpful discussions. We are also grateful to the helpful comments from Alexander J. Hartemink, Yusu

Wang, Hai Yu, and the anonymous reviewers. This work was supported by NSF under ITR Grant EIA-0086015 and ITR Grant 0326157, by NSF under QuBIC Grant EIA-0218376 and QuBIC Grant EIA-0218359, by NSF under EMT Grant CCF-0432038 and EMT Grant CCF-0432047, by DARPA/AFSOR under Contract F30602-01-2-0561, and by RGC under Grant HKBU2107/04E.

References

1. P. Alberti and J. L. Mergny. DNA duplex-quadruplex exchange as the basis for a nanomolecular machine. *Proc. Natl. Acad. Sci. USA*, 100:1569–1573, 2003.
2. Y. Benenson, R. Adar, T. Paz-Elizur, Z. Livneh, and E. Shapiro. DNA molecule provides a computing machine with both data and fuel. *Proc. Natl. Acad. Sci. USA*, 100:2191–2196, 2003.
3. Y. Chen, M. Wang, and C. Mao. An autonomous DNA nanomotor powered by a DNA enzyme. *Angew. Chem. Int. Ed.*, 43:3554–3557, 2004.
4. L. Feng, S. H. Park, J. H. Reif, and H. Yan. A two-state DNA lattice switched by DNA nanoactuator. *Angew. Chem. Int. Ed.*, 42:4342–4346, 2003.
5. T. H. LaBean, H. Yan, J. Kopatsch, F. Liu, E. Winfree, J. H. Reif, and N. C. Seeman. The construction, analysis, ligation and self-assembly of DNA triple crossover complexes. *J. Am. Chem. Soc.*, 122:1848–1860, 2000.
6. J. Li and W. Tan. A single DNA molecule nanomotor. *Nano Lett.*, 2:315–318, 2002.
7. D. Liu and S. Balasubramanian. A proton fuelled DNA nanomachine. *Angew. Chem. Int. Ed.*, 42:5734–5736, 2003.
8. C. Mao, W. Sun, and N. C. Seeman. Designed two-dimensional DNA holliday junction arrays visualized by atomic force microscopy. *J. Am. Chem. Soc.*, 121:5437–5443, 1999.
9. C. Mao, W. Sun, Z. Shen, and N. C. Seeman. A DNA nanomechanical device based on the B-Z transition. *Nature*, 397:144–146, 1999.
10. J. H. Reif. The design of autonomous DNA nanomechanical devices: Walking and rolling DNA. *Lecture Notes in Computer Science*, 2568:22–37, 2003. Published in Natural Computing, DNA8 special issue, Vol. 2, p 439-461, (2003).
11. N. C. Seeman. DNA in a material world. *Nature*, 421:427–431, 2003.
12. W. B. Sherman and N. C. Seeman. A precisely controlled DNA biped walking device. *Nano Lett.*, 4:1203–1207, 2004.
13. F. C. Simmel and B. Yurke. Using DNA to construct and power a nanoactuator. *Phys. Rev. E*, 63:041913, 2001.
14. F. C. Simmel and B. Yurke. A DNA-based molecular device switchable between three distinct mechanical states. *Appl. Phys. Lett.*, 80:883–885, 2002.
15. A. J. Turberfield, J. C. Mitchell, B. Yurke, Jr. A. P. Mills, M. I. Blakey, and F. C. Simmel. DNA fuel for free-running nanomachines. *Phys. Rev. Lett.*, 90:118102, 2003.
16. E. Winfree, F. Liu, L. A. Wenzler, and N. C. Seeman. Design and self-assembly of two-dimensional DNA crystals. *Nature*, 394:539–544, 1998.
17. H. Yan, S. H. Park, G. Finkelstein, J. H. Reif, and T. H. LaBean. DNA-templated self-assembly of protein arrays and highly conductive nanowires. *Science*, 301:1882–1884, 2003.
18. H. Yan, X. Zhang, Z. Shen, and N. C. Seeman. A robust DNA mechanical device controlled by hybridization topology. *Nature*, 415:62–65, 2002.

19. P. Yin, A. J. Turberfield, and J. H. Reif. Designs of autonomous unidirectional walking DNA devices. Technical Report CS-2004-01, Duke University, Computer Science Department, 2004.
20. P. Yin, H. Yan, X. G. Daniell, A. J. Turberfield, and J. H. Reif. A unidirectional DNA walker moving autonomously along a linear track. *Angew. Chem. Int. Ed.*, 2004. In press.
21. B. Yurke, A. J. Turberfield, Jr. A. P. Mills, F. C. Simmel, and J. L. Neumann. A DNA-fuelled molecular machine made of DNA. *Nature*, 406:605–608, 2000.

Design of an Autonomous DNA Nanomechanical Device Capable of Universal Computation and Universal Translational Motion[⋆]

Peng Yin[1], Andrew J. Turberfield[2], Sudheer Sahu[1], and John H. Reif[1]

[1] Department of Computer Science, Duke University,
Box 90129, Durham, NC 27708-0129, USA
{py, sudheer, reif}@cs.duke.edu

[2] University of Oxford, Department of Physics, Clarendon Laboratory,
Parks Road, Oxford OX 1 3PU, UK
a.turberfield@physics.ox.ac.uk

Abstract. Intelligent nanomechanical devices that operate in an autonomous fashion are of great theoretical and practical interest. Recent successes in building large scale DNA nano-structures, in constructing DNA mechanical devices, and in DNA computing provide a solid foundation for the next step forward: designing autonomous DNA mechanical devices capable of arbitrarily complex behavior. One prototype system towards this goal can be an autonomous DNA mechanical device capable of universal computation, by mimicking the operation of a universal Turing machine. Building on our prior theoretical design and prototype experimental construction of an autonomous unidirectional DNA walking device moving along a linear track, we present here the design of a nanomechanical DNA device that autonomously mimics the operation of a 2-state 5-color universal Turing machine. Our autonomous nanomechanical device, called an Autonomous DNA Turing Machine (ADTM), is thus capable of universal computation and hence complex translational motion, which we define as *universal translational motion*.

1 Introduction

1.1 Previous and Current Work

DNA has been explored as an excellent material for building large scale nano-structures, constructing individual nanomechanical devices, and performing computations [26]. Recent years have seen substantial progress in these three fields and this progress provides a solid foundation for the next step forward: designing (and implementing) autonomous nanomechanical devices capable of arbitrarily complex behavior, e.g. motion and computation. A major challenge in nanoscience is to design a nanomechanical device that is capable of *universal translational motion*, which we define as the motion determined by the head of a universal Turing machine. To meet this challenge, we describe the construction of a nanomechanical device embedded in a DNA lattice that

[⋆] Extended abstract. For full version, see [41].

mimics the operation of a universal Turing machine in an autonomous fashion. We call this prototype device an Autonomous DNA Turing Machine, or ADTM. The design of ADTM benefits from recent progress in the three aforementioned fields, can be viewed as an exciting synergistic point of the three fields, and in turn contributes to the advance of each of these fields: it can be viewed as an autonomous intelligent nano-lattice, an autonomous intelligent nanorobotics device, and a compact autonomous universal computing device.

Large scale periodic lattices have been made from a diverse set of branched DNA molecules, such as the DX molecules [35], TX molecules [10], rhombus molecules [17], and 4x4 molecules [38]. In addition, researchers have also successfully constructed aperiodic programmable DNA lattices [37]. These self-assembled lattices provide a structural base for our construction of ADTM, whose main structure can be implemented as two parallel arrays of DNA molecules embedded in a one-dimensional DNA lattice.

A variety of DNA nanomechanical devices have been previously constructed that demonstrate motions such as open/close [14, 28, 29, 43], extension/contraction [2, 8, 12], and rotation [18, 39], mediated by external environmental changes such as the addition and removal of DNA fuel strands [2, 8, 12, 28, 29, 39, 43] or the change of ionic composition of the solution [14, 18]. A desirable improvement of these devices is the construction of DNA nanomechanical devices that achieve motions beyond the non-autonomous and localized non-extensible motions exhibited by the above devices. There have already been some exciting work in this direction. Turberfield and colleagues have designed a free running DNA machine [31] which uses DNA as fuels and demonstrates autonomous motion. Mao's group recently demonstrated an autonomous DNA motor powered by a DNA enzyme [6]. Seeman's group has constructed a DNA walking device controlled by DNA fuel strands [27]. Reif has designed an autonomous DNA walking device and an autonomous DNA rolling device that move in a random bidirectional fashion along DNA tracks [23]. The authors have recently designed autonomous DNA walking devices capable of autonomous programmable unidirectional motions along linear tracks [40, 42]. One of these prototype unidirectional DNA walking devices has also been experimentally constructed in our lab [42]. However, these devices only exhibit simple unidirectional motion.

A rich family of DNA computation schemes have been proposed and implemented [7, 11, 13, 15, 16, 19, 21, 22, 20, 24, 25, 30, 34] following Adleman's seminal report in 1994 [1]. Among them, the most relevant work that has inspired the construction here is the universal DNA Turing machine design by Rothemund [24] and the autonomous 2-state 2-color finite state automata constructed by Shapiro's group [3, 4, 5]. In Rothemund's innovative design, the transition table of a universal Turing machine is encoded in a circular DNA and the encoded transitions are carried out by enzymic cleavages and ligations. However, these reactions need to be carried out manually for each transition. In contrast, the ADTM described here operates in an autonomous fashion with no external environmental mediation. In the inspiring construction by Shapiro's group, a duplex DNA encoding the sequence of input symbols is digested sequentially by an endonuclease in a fashion mimicking the processing of input data by a finite state automaton. Some of their encoding schemes are used in the construction described in this paper, i.e. using DNA sticky end to encode the combination of state and symbol and using a class

II restriction enzyme to effect state change. A limitation of the finite state automata construction is that the data are destroyed as the finite state automaton proceeds. Though this feature does not affect the proper operation of a finite state automaton, it poses a barrier to further extending the finite state automaton to more powerful computing devices such as Turing machines.

In this work, we encode computational power into a DNA walking device embedded in a DNA lattice and thus accomplish the design for an autonomous nanomechanical device capable of universal computation, by mimicking the operation of a 2-state 5-color universal Turing machine. In the process of computation, the device can also demonstrate universal translational motion as defined above, i.e. the motion demonstrated by the head of universal Turing machine.

1.2 Universal Turing Machine

A Turing machine is a theoretical computational device invented by Turing for performing mechanical or algorithmic mathematical calculations [32, 33]. Though the construction and operational rules of a Turing machine may seem beguilingly simple and rudimentary, it has been shown that any computational process that can be done by present computers can be carried out by a Turing machine.

A Turing machine consists of two parts, a *read-write head* and a linear tape of *cells* encoding the input data. The head has an internal state q and each cell has a color (or data) c (as described in [36]). At any step, the head resides on top of one cell, and the color of that cell and the state of the head together determines a transition: (i) the current cell may change to another color; (ii) the head may take a new state; (iii) the head may move to the cell immediately to the left or the right of the current cell.

A universal Turing machine is a Turing machine that can simulate the operation of any other Turing machine. Let m and n be the number of possible states and the number of possible colors of a Turing machine, respectively. The Turing machine with a proven universal computation capacity and the smallest $m \times n$ value is a 2-state 5-color Turing machine described in [36]. In this paper, we describe the design of a DNA nanomechanical device that simulates the general operation of an arbitrary 2-state 5-color Turing machine whose head moves to either its left or right neighbor in every transition, and in particular, the universal Turing machine described in [36].

The rest of the paper is organized as follows. We give a structural overview of ADTM in Sect. 2 and an operational overview in Sect. 3, followed by a detailed step-by-step molecular implementation of the operation in Sect. 4. In Sect. 5, we briefly overview two major technical challenges in designing ADTM. We close in Sect. 6 with a discussion of our results.

2 Structural Overview

Figure 1 illustrates the structure of ADTM. ADTM operates in a solution system. The major components of ADTM are two parallel arrays of *dangling molecules* tethered to two *rigid* tracks. The two rigid tracks can be implemented as rigid DNA lattices as described in Sect. 1.1, for example, the rhombus lattice [17] as shown in Figure 1. A

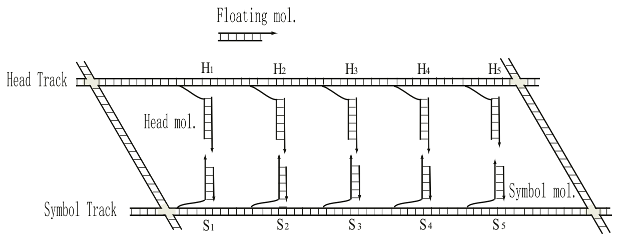


Fig. 1. Schematic drawing of the structure of ADTM. H_i and S_i denote Head-Molecule and Symbol-Molecule, respectively. The backbones of DNA strands are depicted as line segments. The short bars represent base pairing between DNA strands

dangling-molecule is a duplex DNA fragment with one end linked to the track via a flexible single strand DNA and the other end possessing a single strand DNA extension (the sticky end). Due to the flexibility of the single strand DNA linkage, a dangling-molecule moves rather freely around its joint at the track. The upper and lower arrays of dangling-molecules are called *Head-Molecules*, denoted as H, and *Symbol-Molecules*, denoted as S, respectively. We require that the only possible interactions between two dangling-molecules are either a reaction between a Head-Molecule and the Symbol-Molecule immediately below it or a reaction between two neighboring dangling-molecules along the same track. This requirement can be ensured by the rigidity of the tracks and the properly spacing of dangling-molecules along the rigid tracks.

In addition to the two arrays of dangling-molecules, there are *floating-molecules*. A floating-molecule is a free floating (unattached to the tracks) duplex DNA segment with a single strand overhang at one end (sticky end). A floating-molecule floats freely in the solution and thus can interact with another floating-molecule or a dangling-molecule provided that they possess complementary sticky ends. There are two kinds of floating-molecules: the Rule-Molecules and the Assisting-Molecules. The Rule-Molecules specify the computational rules and are the programmable part of ADTM while the Assisting-Molecules assist in the carrying out the operations of ADTM, as described in detail later.

The array of Symbol-Molecules represent the data tape of a Turing Machine; the array of Head-Molecules represent the moving head of a Turing Machine (more specifically, at any time, only one Head-Molecule is active, and its position indicates the position of the head of a Turing Machine); the Rule-Molecules collectively specify the transition rules for ADTM; the Assisting-Molecules are auxiliary molecules that assist in maintaining the operation of ADTM.

The duplex portion and/or the sticky end of a DNA molecule may encode the following information: (i) *state*, the Turing machine state; (ii) *color*, the color (data) encoded in a symbol molecule; (iii) *position*, the position type of a Head-Molecule. The state, color, and position information are denoted as q, c, and p, respectively, where $q \in \{Q_A = LONG, Q_B = SHORT\}$, $c \in \{C_A, C_B, C_C, C_D, C_E\}$, $p \in \{P_A, P_B, P_C\}$ for the 2-state 5-color ADTM. The position information p indicates the position type of a Head-Molecule. This information is essential for dictating the bidirectional motion of the head. An information encoding molecule is denoted as $X^a[y]^b$, where X is its duplex portion, $[y]$ is its sticky end portion; a is the state/color/position information encoded in X, and b is the state/color/position information encoded in $[y]$. A complementary sticky end of $[y]$ is denoted as $[\bar{y}]$.

The array of Head-Molecules is denoted as $(H_1, H_2, H_3, \dots)$; the array of Symbol-Molecules is denoted as $(S_1, S_2, S_3, \dots)$. To specify the motion of ADTM head, we have Head-Molecules arranged in periodic linear order along the Head-track

$$(H_1^{P_A}, H_2^{P_B}, H_3^{P_C}, H_4^{P_A}, H_5^{P_B}, H_6^{P_C} \dots)$$

3 Operational Overview

At the beginning of a transition operation of ADTM, all the Symbol-Molecules possess sticky ends $[\bar{s}]$. A Symbol-Molecule with a sticky end $[\bar{s}]$ is referred to as in its *default* configuration; the $[\bar{s}]$ sticky end is referred to as a *default sticky end*. One of the Head-Molecules encodes the current state of ADTM in its duplex portion and possesses an *active* sticky end $[s]$ that is complementary to the sticky end $[\bar{s}]$ of the Symbol-Molecule just below it. This Head-Molecule is referred to as the *active Head-Molecule*. In contrast, all other Head-Molecules (with sticky ends other than $[s]$) are in *default* or *inactive* configuration.

Figure 2 gives a high level description of the events that occur during one transition of ADTM. For ease of exposition, we describe the operation in 4 stages. The 8 types of ligation events that correspond to the detailed 8-step implementation of ADTM (Sect. 4) are also marked in the figure to assist the reader in relating the high level description in this section to detailed step-by-step implementation in Sect. 4. In Stage 1, the active Head-Molecule (labeled with a triangle, H_3 in the example shown in Figure 2) is ligated to the Symbol-Molecule (S_3 in Figure 2) directly below it, creating an endonuclease recognition site in the ligation product (event (1) in Figure 2). The ligation product is subsequently cleaved into two molecules by an endonuclease. The sticky end of each of the two newly generated molecules encodes the current state and the current color of ADTM.

In Stage 2, both the new Symbol-Molecule and the new Head-Molecule are ligated to floating Rule-Molecules (events (2) and (4) in Figure 2), which possess complementary sticky ends to them and correspond to one entry in the Turing machine transition table. The ligation product between the Symbol-Molecule and the Rule-Molecule is in turn cleaved, generating a new Symbol-Molecule dictated by the current state and color information as well as the transition rule. The new Symbol-Molecule encodes the new color in its sticky end. Similarly, the ligation product between the Head-Molecule and the Rule-Molecule is cleaved, generating a new Head-Molecule whose duplex portion encodes information of Turing machine's next state and whose sticky end encodes the moving direction of the head.

In Stage 3, the newly generated Symbol-Molecule is further modified by an Assisting-Molecule so that it will encode the new color in its duplex portion (rather than sticky end) and possess an $[\bar{s}]$ sticky end (event (3) in Figure 2). The sticky end of the Head-Molecule will dictate it to hybridize with either the Head-Molecule to its left or to its right, depending on which of its neighbors possesses a complementary sticky end (event (5) in Figure 2, H_3 is ligated with its left neighbor H_2). Next, the ligation product between these two Head-Molecules is cleaved.

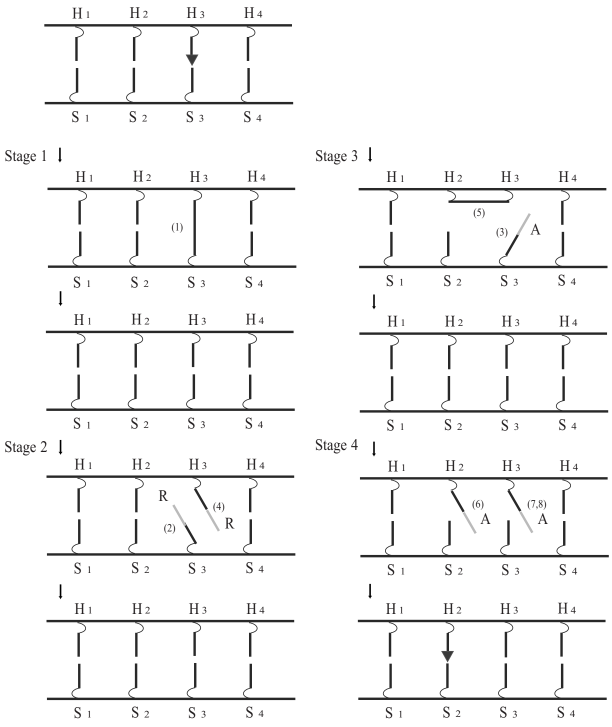


Fig. 2. Operational overview of ADTM. The dangling Head-Molecules and Symbol-Molecules are depicted as dark line fragments. The floating Rule-Molecules and Assisting-Molecules are depicted as light colored light segments. H, S, R, and A denote Head-Molecule, Symbol-Molecule, Rule-Molecule, and Assisting-Molecule, respectively. The triangle indicates the *active* Head-Molecule. (i) indicates the ligation event that occurs in Step i as will be described in the detailed step-by-step implementation of ADTM (Sect. 4)

In Stage 4, the two Head-Molecules are modified by floating Assisting-Molecules (events (6) and $(7, 8)$ in Figure 2) so that the first Head-Molecule is restored to its inactive configuration (with a default sticky end) and the second Head-Molecule encodes the state information in its duplex part and possesses an active sticky end $[s]$ and thus becomes an active Head-Molecule, ready to interact with the Symbol-Molecule located directly below it.

This finishes a transition and the operation can thus go on inductively. We emphasize that we describe the events in stages only for ease of exposition. The proper operation of ADTM does not require the synchronization of the events as described above. For example, event (3) in Stage 3 can occur either before event (4) or after event (8).

4 Step-by-Step Implementation

We next give a detailed 8-step description of the operation of ADTM. Each step consists of ligation and cleavage events. The ligation events are marked in Figure 2 with (i), where $i = 1, 2, \ldots, 8$. To demonstrate the practicality of our design, we give full DNA sequence for the reactions of each step. In addition to the A, T, C, and G bases, we also occasionally require another pair of unnatural bases which we denote as E and F. The reason to use E and F is to minimize the futile reactions as described later and hence increase the efficiency of our ADTM. The practicality of use of E and F is justified by the existing technology to make such bases and incorporate them into DNA strands. For a recent survey on unnatural bases, see [9].

At the start of the operation of ADTM, the configuration of the Head-Molecules array along the Head-track is

$$(\hat{H}_1^{p_1 q}[s])([\bar{h}]^{p_2} H_2^{p_2})([\bar{h}]^{p_3} H_3^{p_3})\ldots$$

where $p_i = P_A$ for $i = 3k+1$, $p_i = P_B$ for $i = 3k+2$, $p_i = P_C$ for $i = 3k+3$ for $k = 0, 1, 2, \ldots$. The first Head-Molecule is special: it is the active Head-Molecule and represents the current position of the active head. We use the symbol ˆ to denote the active configuration of a Head-Molecule. H_1 has the unique sticky end $[s]$, which is complementary to the sticky end $[\bar{s}]$ of a Symbol-Molecule in default configuration (in particular, the Symbol-Molecule directly below it). Thus, H_1 can hybridize and be ligated with Symbol-Molecule S_1, and this will start the operation of the Turing machine. Recall that p encodes the position type information of a Head-Molecule. This position type information is encoded both in the sticky end portion and in the duplex portion of a Head-Molecule. As we will see below, the sticky end encoding of p is necessary for dictating the appropriate motion of an active head; the duplex portion encoding is necessary for restoring a Head-Molecule to its default configuration after it turns from an active to an inactive state.

The Symbol-Molecules array along the Symbol-track is

$$([\bar{s}]S_1^{c_1})([\bar{s}]S_2^{c_2})([\bar{s}]S_3^{c_3})\ldots$$

All the Symbol-Molecules have the same sticky end $[\bar{s}]$. As such, whenever a Head-Molecule directly above a Symbol-Molecule becomes active, this Symbol-Molecule can interact with the active Head-Molecule. Note that $[\bar{s}]$ encodes no color information – the color information c_i is instead encoded completely in the duplex portion of a Symbol-Molecule.

4.1 Reaction Between a Head-Molecule and a Symbol-Molecule

Step 1. In step 1, the active state-encoding Head-Molecule is first ligated with the color-encoding Symbol-Molecule below it, and then the ligation product is cut into a new Head-Molecule and a new Symbol-Molecule, the sticky ends of which both encode the current state and color information.

Let $\hat{H}_i^{pq}[s]$ be the current active head (encoding position type p and current state q); let $[\bar{s}]S_i^c$ be the Symbol-Molecule below it (encoding current color c). $\hat{H}_i$ and S_i

has complementary sticky ends and hence these two are ligated into $(H_iS_i)^{pqc}$. An endonuclease recognizes the newly formed recognition site in the ligation product and cuts the ligation product into $\hat{H}_i^p[r]^{qc}$ and $[\bar{r}]^{qc}S_i$. Now the sticky ends of both $\hat{H}_i$ and S_i encode the current color and state. Step 1 can be described by the following equation,

$$\hat{H}_i^{pq}[s] + [\bar{s}]S_i^c \rightarrow (H_iS_i)^{pqc} \rightarrow \hat{H}_i^p[r]^{qc} + [\bar{r}]^{qc}S_i$$

The first part of the equation is the ligation of Head-Molecule $\hat{H}_i^{pq}[s]$ with Symbol-Molecule $[\bar{s}]S_i^c$ into $(H_iS_i)^{pqc}$; the second part is the cleavage of the ligation product into Head-Molecule $\hat{H}_i^p[r]^{qc}$ and Symbol-Molecule $[\bar{r}]^{qc}S_i$. Note that now both the sticky ends of the Head-Molecule and the Symbol-Molecule are encoding the current state and color. This encoding scheme is in the same spirit as the one used in [5].

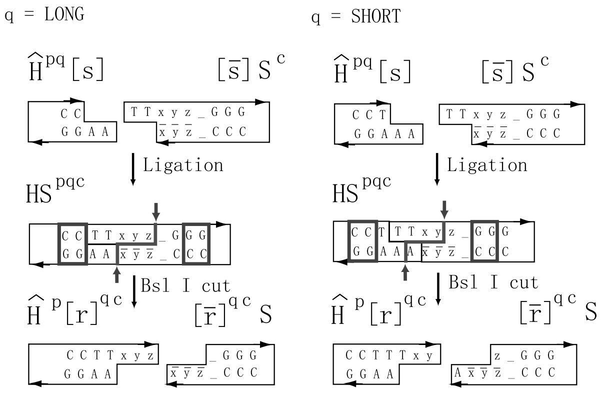


Fig. 3. Step 1 of the operation of ADTM. The current state q and color c are initially encoded in the duplex portion of the Head-Molecule and the Symbol-Molecule, respectively. After the ligation and cleavage, both the sticky ends of the new Head-Molecule and Symbol-Molecule encode the current state q and the current color c. The encoding scheme of c is described in Table 1. Bsl I recognition sites and cleavage sites are indicated with boxes and pairs of bold arrows, respectively

Figure 3 gives the molecular implementation of Step 1. For simplicity, only the relevant end sequences are given. The encoded information p is not shown. Both the case when $q = SHORT$ and the case when $q = LONG$ are depicted. xyz is the color encoding region for Symbol-Molecule S. The encoding scheme used is shown in Table 1.

Table 1. The molecular implementation of the color encoding scheme of a Symbol-Molecule. c is the color; xyz is the sticky $[r]$ exposed when state $q = LONG$; Txy is the sticky end $[r]$ when state $q = SHORT$. Note that all the ten sticky end sequences are different from each other

	c	C_A	C_B	C_C	C_D	C_E
$q = LONG$	xyz	TTA	CTT	CAA	AEA	CEA
$q = SHORT$	Txy	TTT	TCT	TCA	TAE	TCE

4.2 Color Change of a Symbol-Molecule

After Step 1, the sticky end of $[\bar{r}]^{qc}S_i$ encodes the current state and color. This sticky end is subsequently detected by a Rule-Molecule $\tilde{R}[r]^{qc}$, which has a complementary sticky end. $\tilde{R}[r]^{qc}$ corresponds to one entry in the transition table for ADTM, and determines the next color c' that will be encoded in S_i. This color transition occurs in Step 2 and S_i is modified to possess a sticky end $[\bar{e}]^{c'}$ that encodes the new color c'. In Step 3, S_i is restored to a default configuration with a sticky end $[\bar{s}]$, and the new color c' encoded in its duplex portion. We next describe the reactions in detail.

Step 2. In Step 2, Rule-Molecule $\tilde{R}[r]^{qc}$ hybridizes and is ligated with Symbol-Molecule $[\bar{r}]^{qc}S_i$. The ligation product is cut into $\tilde{R}_w[e]^{c'}$ (a waste molecule that diffuses away) and $[\bar{e}]^{c'}S_i$. The sticky end $[\bar{e}]$ encodes the new color c'. Schematically, we have,

$$\tilde{R}[r]^{qc} + [\bar{r}]^{qc}S_i \rightarrow (RS_i)^{qcc'} \rightarrow \tilde{R}_w^{qc}[e]^{c'} + [\bar{e}]^{c'}S_i$$

Figure 4 describes the molecular implementation of Step 2 for the case when current state is $q = LONG$, and the new color is $c' = C_B$. The case for $q = SHORT$ is similar, except that sticky end $[\bar{r}]$ of S is $C\bar{x}\bar{y}$ instead of $\bar{x}\bar{y}\bar{z}$. The Rule-Molecule $\tilde{R}[r]^{qc}$ consists of three parts, in the terminology of [5], Bpm I recognition site, spacer region, and <state,color> detector. The <state, color> detector is the sticky end $[r]^{qc}$, which hybridizes with and thus detects the sticky end $[\bar{r}]^{qc}$ of the symbol molecule. The Rule-Molecule and the Symbol-Molecule are ligated and Bpm I cuts the ligation product into a waste Rule-Molecule $\tilde{R}_w^{qc}[e]^{c'}$ ($_w$ for waste), which diffuses away, and a new Symbol-Molecule $[\bar{e}]^{c'}S$, effecting the color change of the Symbol-Molecule from c to c'. The length of the spacer of $\tilde{R}$ (see Figure 4) determines the position of the cut in the ligation product and hence the sticky end $[\bar{e}]$ and the new color c' encoded in it. See Table 2 for the relation between the length of the spacer, the sequence of sticky end $[\bar{e}]$ and the new color c'.

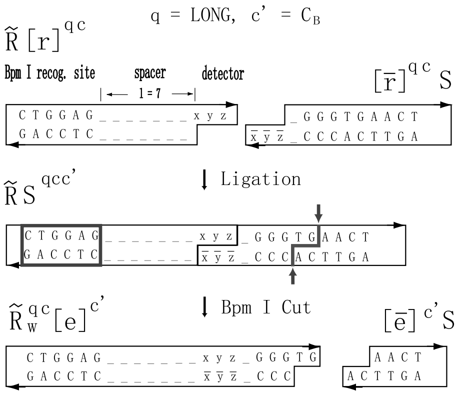


Fig. 4. Step 2 of the operation of ADTM. In this example, the current state is $q = LONG$; the current color q and state c are encoded in the Symbol-Molecule's sticky end $[\bar{r}]$ whose sequence is $\bar{x}\bar{y}\bar{z}$; the new color, in this case, will be $c' = C_B$, encoded in sticky end $[\bar{e}]$ whose sequence is "TG" Bpm I recognition site and cleavage site are indicated with a box and a pair of bold arrows, respectively

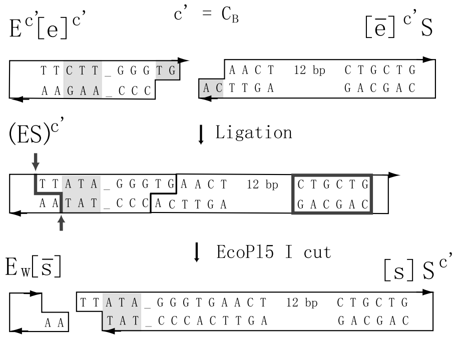
Table 2. The relation between the length of the spacer, the sequence of sticky end $[\bar{e}]$ and the new color of a Symbol-Molecule. l is the spacer length; $\bar{e}$ is the sticky end sequence; c' is the new color

	c'	C_A	C_B	C_C	C_D	C_E
	$\bar{e}$	CA	AC	CT	TT	TG
$q = LONG$	l	8	7	6	5	4
$q = SHORT$	l	7	6	5	4	3

Step 3. The Symbol-Molecule $[\bar{e}]^{c'}S_i$ obtained from Step 2 needs to be restored to its default configuration $[\bar{s}]S_i^{c'}$ so that it can interact with the Head-Molecule H_i above it when H_i becomes active again. Note that this re-usability of the Symbol-Molecule is essential for the proper functioning of ADTM. After the restoration, the new color c' is encoded in the duplex portion of S_i, whose sticky end is the default sticky end $[\bar{s}]$ for a Symbol-Molecule: it encodes no color, but is ready to interact with an active Head-Molecule. The reaction of Step 3 is,

$$E^{c'}[e]^{c'} + [\bar{e}]^{c'}S_i \rightarrow (ES)^{c'} \rightarrow E_w[s] + [\bar{s}]S_i^{c'}$$

Figure 5 gives a molecular implementation of Step 3 for the case $c' = C_B$. Color c' is encoded both in the sticky end portion and the duplex portion of Assisting-Molecule $E^{c'}[e]^{c'}$. Assisting-Molecule $E^{c'}[e]^{c'}$ detects the color encoding sticky end of Symbol-Molecule S_i and transfers its color encoding duplex portion to S_i via ligation and subsequent cleavage. This step generates a waste product $E_w[s]$ that diffuses away. Note that $E_w[s]$ may hybridize and be ligated with some other $[\bar{s}]$ end of a Symbol-Molecules,say $[\bar{s}]S_k$. However, this only represents some futile reactions that will not block, reverse, or alter the operation of ADTM, since $E_w[s]$ will be cut subsequently away from $[\bar{s}]S_k$ by EcoPl 5I. Nevertheless, this does decrease the efficiency of ADTM and as the concentration of $E_w[s]$ increases, the negative effect on the efficiency becomes more prominent. For a complete set of Assisting-Molecules $E^{c'}[e]^{c'}$, see Figure 6.



E^c'[e]^c'
c' = C_B
[ē]^c' S
T T C T T _ G G G T G
A A G A A _ C C C
A A C T 12 bp C T G C T G
A C T T G A G A C G A C
(ES)^c'
Ligation
T T A T A _ G G G T G A A C T 12 bp C T G C T G
A A T A T _ C C C A C T T G A G A C G A C
EcoP15 I cut
E_w[s̄]
[s] S^c'
A A
T T A T A _ G G G T G A A C T 12 bp C T G C T G
T A T _ C C C A C T T G A G A C G A C

Fig. 5. Step 3 of the operation of ADTM. In this example, the new color $c' = C_B$. See Figure 6 for the complete set of Assisting-Molecules $E^{c'}e^{c'}$. The color encoding regions are indicated with light gray background. EcoPl5 I recognition site and cleavage site are indicated with a box and a pair of bold arrows, respectively

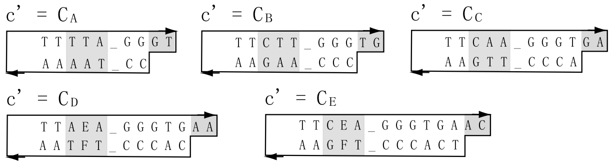


Fig. 6. The complete set of Assisting-Molecules $E^{c'}[e]^{c'}$. The color encoding regions are indicated with light gray background

Note that the existence of endonuclease EcoPl5 I recognition site in the duplex portion of S_i adds extra complication to Step 1 and Step 2: it results in futile reactions which are discussed in [41].

4.3 State Change of a Head-Molecule

Step 4. In Step 4, the Head-Molecule $H_i^p[r]^{qc}$ generated in Step 2 (with its sticky end encoding the current state and color) is modified by a Rule-Molecule that decides the state transition and the motion of the head. After the modification, the new state information is encoded in the duplex portion of the modified Head-Molecule, and the motion direction of the head is encoded in the sticky end of the modified Head-Molecule in the form of a sticky end complementary to one of its neighboring Head-Molecules. The sticky end of the modified Head-Molecule will dictate it to interact with either its left or right neighbors, and thus determines the motion of the head.

More specifically, Head-Molecule $H_i^p[r]^{qc}$ hybridizes and is ligated with a free floating Rule-Molecule $[\bar{r}]^{qc}R$ and the ligation product $(H_iR)^{pqc}$ is cut by endonuclease EcoPl5 I into $H_i^{pq'}[h]^{p'}$ and $[\bar{h}]^{p'}R_w$, a waste molecule that diffuses away. Head-Molecule $H_i^{pq'}[h]^{p'}$ encodes the new state q' in its duplex portion, and the motion direction p' of the head in its sticky end. The reaction of Step 4 is,

$$\hat{H_i}^p[r]^{qc} + [\bar{r}]^{qc}R \rightarrow (H_iR)^{pqc} \rightarrow \hat{H_i}^{pq'}[h]^{p'} + [\bar{h}]^{p'}R_w$$

Figure 7 describes the molecular implementation for the case when the current state $q = LONG$; new state $q' = SHORT$; the position type of the current Head-Molecule H_i is $p = P_A$; the position type of the Head-Molecule H_j that it will interact with is $p' = P_B$ (hence $j = i + 1$ in this case). The Rule-Molecule $[\bar{r}]^{qc}R$ consists of three parts: the detector sticky end $[\bar{r}]^{qc}$ that encodes the current state and color; the spacer, whose length determines the transition results (new state and motion direction of the head); and recognition site for endonuclease EcoPl5 I. The Rule-Molecule $[\bar{r}]^{qc}R$ detects the current state q and color c encoded in sticky end $[r]^{qc}$ of H_i and is ligated to H_i. After ligation, endonuclease EcoPl5 I cuts into the motion encoding region of the Head-Molecule and exposes a new sticky end that encodes the position type information p' (and hence determines the motion direction). Cleavages at motion encoding regions I and II result in new states $q' = LONG$ and $q' = SHORT$, respectively. The complete set of transitions for all the combinations of different p, q, d, and q' is described in [41].

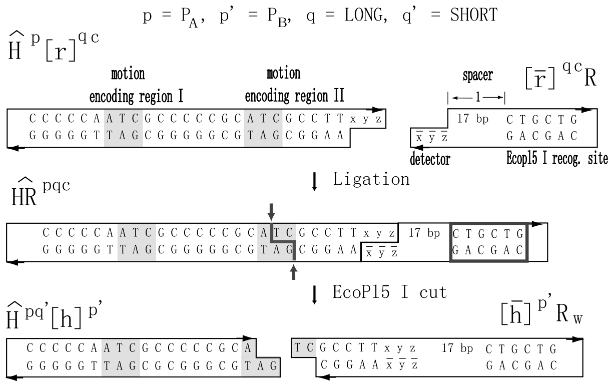


Fig. 7. Step 4 of the operation of ADTM. The motion encoding regions are indicated with light gray background. l is the length of the spacer region of Rule-Molecule R. EcoPl5 I recognition site and cleavage site are indicated with a box and a pair of bold arrows, respectively

4.4 Reaction Between Two Adjacent Head-Molecules

Head-Molecule $\hat{H}_i^{pq'}[h]^{p'}$ produced in Step 4 will next interact with one of its neighboring Head-Molecules, $[\bar{h}]^{p'}H'_j$, where $j = i - 1$ for its left neighbor and $j = i + 1$ for its right neighbor (Step 5). Then H_j becomes an active Head-Molecule encoding the new state q' (Step 6) while H_i is restored to its default inactive configuration (Steps 7 and 8).

Step 5. In Step 5, Head-Molecule $\hat{H}_i^{pq'}[h]^{p'}$ is ligated to either its left neighbor or its right neighbor $[\bar{h}]^{p'}H'_j$, where $j = i - 1$ or $i + 1$, as dictated by the p' information encoded in its sticky end. The ligation product $(H_iH_j)^{q'}$ is cut into $H_i^p[\bar{t}]^{pp'q'}$ and $[t]^{pp'q'}\hat{H}_j$. The reaction of Step 5 is,

$$\hat{H}_i^{pq'}[h]^{p'} + [\bar{h}]^{p'}H'_j \rightarrow (H_iH_j)^{q'} \rightarrow H_i[\bar{t}]^{pp'q'} + [t]^{pp'q'}\hat{H}'_j$$

Note that now both the sticky ends of H_i and H_j encode position type p of H_i, position type p' of H_j, and the new state q'.

Figure 9 gives a molecular implementation for this step. Panel I depicts an example case in full detail; Panel II and III show all the cases in a simplified way. Note that the sticky end $[\bar{t}]$ (and $[t]$) encodes all the information for position type p of H_i, position type p' of H_j, and the new state q', we hence have $3 \times 2 \times 2 = 12$ different sticky ends $[\bar{t}]$.

Step 6. In Step 6, Head-Molecule $\hat{H}'_j$ is modified into a Head-Molecule ready to interact with a Symbol-Molecule; in other words, it becomes an active head. The reaction of Step 6 is,

$$\hat{H}'_j[t]^{pp'q'} + [\bar{t}]^{pp'q'}T \rightarrow H'_jT \rightarrow \hat{H}_j'^{q'}[s] + [\bar{s}]T_w$$

Figure 8 describes the molecular implementation for Step 6. The mechanism of this step is very similar to Step 4, and hence we omit its details.

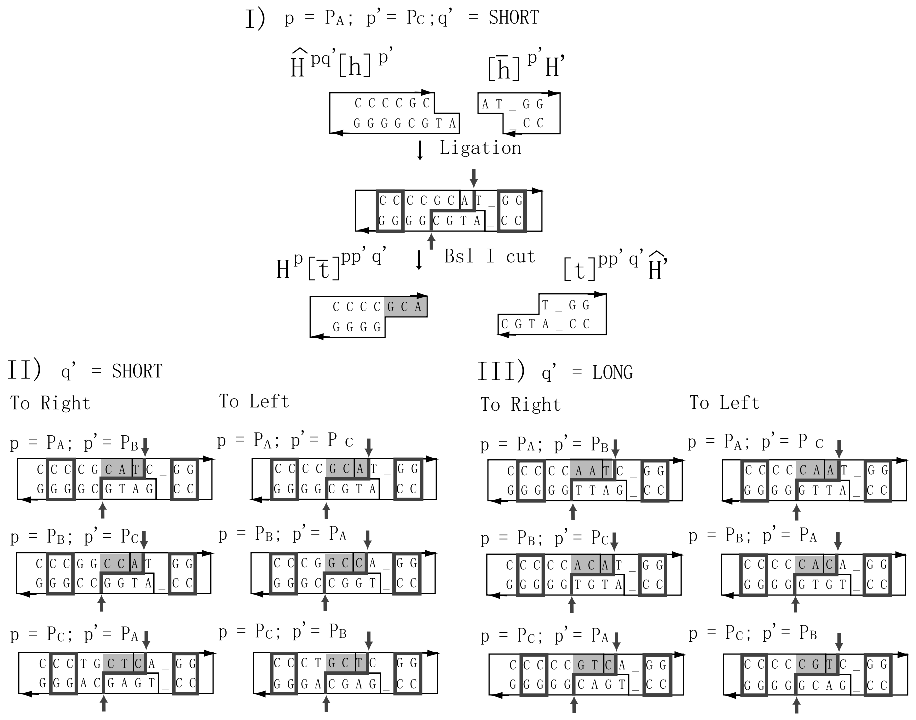


Fig. 8. Step 5 of the operation of ADTM. Panel I depicts the case when $p = P_A$, $p' = P_C$, and $q' = SHORT$. Panel II and III describe all the cases when $q' = SHORT$ and all the cases when $q' = LONG$, respectively. In panel II and III, each case is represented in a simplified fashion that only shows the ligation product before the cleavage.Bsl I recognition sites and cleavage sites are indicated with boxes and pairs of bold arrows, respectively. The unique sticky ends $[\bar{t}]^{pp'q'}$ are shown with gray background

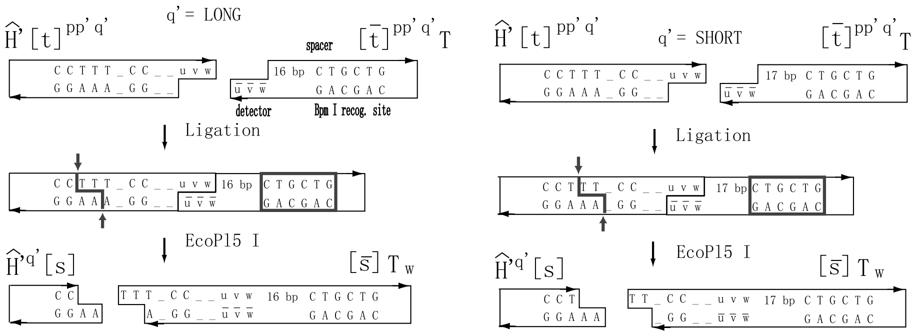


Fig. 9. Step 6 of the operation of ADTM. Bpm I recognition sites and cleavage sites are indicated with boxes and pairs of bold arrows, respectively

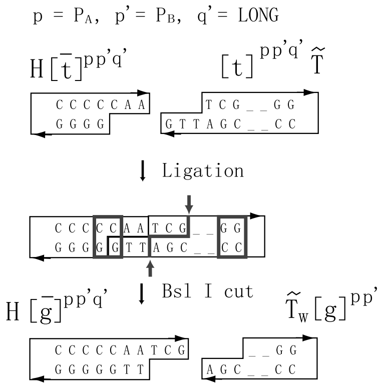


Fig. 10. Step 7 of the operation of ADTM. Bsl I recognition site and cleavage site are indicated with a box and a pair of bold arrows, respectively

Steps 7. and 8. In Step 7, the sticky end $[\bar{t}]^{pp'q'}$ of Head-Molecule H_i is modified by an Assisting-Molecule $\tilde{T}[t]^{pp'q'}$ to a new sticky end $[\bar{g}]^{pp'q'}$. In Step 8, the sticky end $[\bar{g}]^{pp'q'}$ initiates a sequential "growing-back" process which restores H_i to its default (inactive) configuration $[\bar{h}]^p H_i^p$. The reaction of Step 7 is,

$$\tilde{T}[t]^{pp'q'} + [\bar{t}]^{pp'q'} H_i \rightarrow \tilde{T} H_i \rightarrow \tilde{T}_w [g]^{pp'q'} + [\bar{g}]^{pp'q'} H_i$$

The reaction of Step 8 is,

$$[\bar{g}]^{pp'q'} H_i \rightarrow [\bar{h}]^p H_i^p$$

Figure 10 and Figure 11 describe the molecular implementation of Step 7 and Step 8 for the case $p = P_A$, $p' = P_B$, and $q' = LONG$, respectively. The figures are self-explanatory and hence we omit the details for brevity. Note that Step 8 is a rather spectacular process which illustrates a precisely controlled elongation mechanism using alternating ligations and cleavages. This mechanism may be of independent interest for designing other molecular devices.

4.5 Overall Reaction Flow

Putting all the above steps together, we have a schematic drawing for the overall flow of the reactions (Figure 12). The complete molecule set for the construction of our ADTM is described in [41].

To fully test the validity of our complex construction of ADTM, we performed computer simulation of ADTM (See http://www.cs.duke.edu/∼py/paper/dnaUTM/).

5 Technical Challenges

Two major technical challenges in designing ADTM are to accommodate the *futile* reactions occurring during the operation of ADTM and to design ADTM using limited encoding space dictated by the four (six) letter vocabulary of DNA bases and by the sizes of the recognition, restriction, and spacing regions of endonucleases.

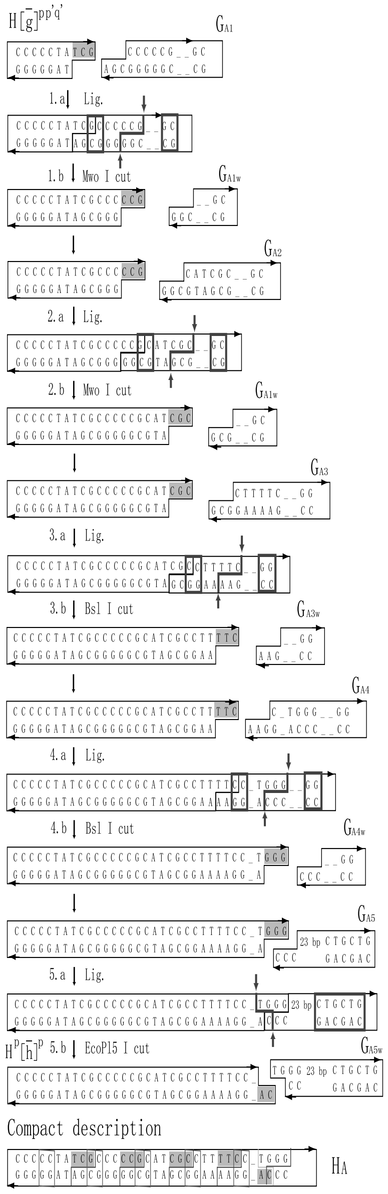


Fig. 11. Step 8 of the operation of ADTM for the case $p = P_A$, $p' = P_B$, and $q' = LONG$. This step consists of a sequence of alternating ligations and cleavages. At each stage k, where $k = 1 - 5$, the Head-Molecule is first ligated to an Assisting-Molecule G_{Ak} (stage $k.a$), then the ligation product is cut by an endonuclease (stage $k.b$). A waste molecule G_{Akw} is generated at each stage. The last panel gives a compact representation of the whole process. The unique sticky end generated at each stage is indicated with gray background. Endonuclease recognition sites and cleavage sites are indicated with boxes and pairs of bold arrows, respectively

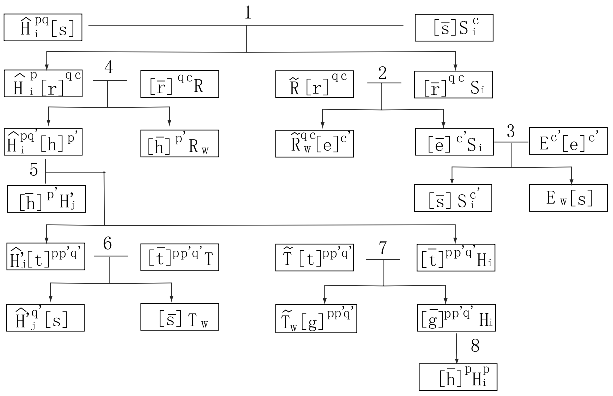


Fig. 12. Overview of the operation of ADTM

The key technique used here to address the first challenge is to make all the futile reactions fully reversible so that they do not obstruct or alter the operation of ADTM. The key technique to address the second challenge is to use an "overlay" technique as shown and to carefully select the sticky ends to avoid undesirable reactions. See [41] for details.

6 Discussion

In this paper, we present the design of a DNA nanomechanical device capable of universal computation and hence universal translational motion. In addition to general design principles, we give detailed molecular implementation of ADTM. A next step would be to construct a DNA cellular automata that demonstrates parallel computations.

As a consequence of the universal computation, ADTM demonstrates universal translational motion. This motion is a symbolic motion in the sense that no physical entity is moved from one location to the other. Instead, the motion is the motion of the active head symbol relative to the tracks. A nanorobotics challenge is to extend ADTM to a device that can move a physical entity, probably a DNA fragment, in a universal translational motion fashion. As a first step, it is conceivable that a DNA nanomechanical device that moves a DNA fragment bidirectionally along the track can be designed and possibly experimentally constructed.

Our complex design of ADTM makes some unconventional physical and chemical assumptions. Two lines of recent work lend partial experimental support to the practicality of this design. The first one is the autonomous DNA finite state automata constructed by Shapiro's group [3, 4, 5], in which a cascade of cleavages and ligations drive the operation of the machine. A more relevant study is the experimental construction of an autonomous unidirectional DNA walker that moves along a DNA track [42]. In this device, a walker moves unidirectionally over a sequence of three dangling anchorages sites (a structural analog to the dangling-molecules in ADTM design) embedded in a DNA track in an autonomous fashion, driven by alternating actions of DNA endonucleases and ligases. In particular, this walking device exploits some very similar enzyme reactions as used in the design of ADTM, such as the ligation and cleavages of DNA duplices tethered to another DNA nanostructure and the ligation of DNA fragments with 3-base overhangs at a relatively high temperature (37 $^\circ C$).

Though a full experimental implementation of ADTM appears daunting, due to the rich set of molecules, reactions, futile reactions involved, it might be possible to experimentally test a subset of the mechanisms described here. Another challenge to experimental demonstration of ADTM is the design of an output detection mechanism.

Many futile reactions happen in the background during the operation of ADTM. A key feature of these futile reactions is that they are *fully reversible*. This is critical in ensuring the autonomous operation of ADTM as explained below. We *initially* supply the system with sufficiently high concentrations of Rule-Molecules and Assisting-Molecules as well as all the *byproducts* generated in the futile reactions. As such, the futile reactions will reach a dynamic balance and the concentrations of all the components involved in the futile reactions, including both the “active” components essential for the operation of ADTM and the “futile” byproducts, will stay relatively constant during the operation of ADTM. Note that since the active components will *not* be depleted by the futile reactions (which could have happened should some futile reactions are irreversible), the autonomous operation of ADTM will not be disrupted. Though the futile reactions are innocuous, they do decrease the efficiency of ADTM. A desirable improvement of the current design is to decrease the level of futile reactions and thus increase the efficiency and robustness of ADTM.

Acknowledgement

The authors would like to thank Hao Yan and Thomas H. LaBean for helpful discussions. We are also grateful to the very helpful comments from Nabil H. Mustafa and the anonymous reviewers. This work was supported by NSF under ITR Grant EIA-0086015 and ITR Grant 0326157, by NSF under QuBIC Grant EIA-0218376 and QuBIC Grant EIA-0218359, by NSF under EMT Grant CCF-0432038 and EMT Grant CCF-0432047, by DARPA/AFSOR under Contract F30602-01-2-0561, and by RGC under Grant HKBU2107/04E.

The full version of this paper and the supplementary material can be accessed at http://www.cs.duke.edu/∼py/paper/dnaUTM/.

References

1. L. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266:1021–1024, 1994.
2. P. Alberti and J. L. Mergny. DNA duplex-quadruplex exchange as the basis for a nanomolecular machine. *Proc. Natl. Acad. Sci. USA*, 100:1569–1573, 2003.
3. Y. Benenson, R. Adar, T. Paz-Elizur, Z. Livneh, and E. Shapiro. DNA molecule provides a computing machine with both data and fuel. *Proc. Natl. Acad. Sci. USA*, 100:2191–2196, 2003.

4. Y. Benenson, B. Gil, U. Ben-Dor, R. Adar, and E. Shapiro. An autonomous molecular computer for logical control of gene expression. *Nature*, 429:423–429, 2004.
5. Y. Benenson, T. Paz-Elizur, R. Adar, E. Keinan, Z. Livneh, and E. Shapiro. Programmable and autonomous computing machine made of biomolecules. *Nature*, 414:430–434, 2001.
6. Y. Chen, M. Wang, and C. Mao. An autonomous DNA nanomotor powered by a DNA enzyme. *Angew. Chem. Int. Ed.*, 43:3554–3557, 2004.
7. D. Faulhammer, A. R. Cukras, R. J. Lipton, and L. F. Landweber. Molecular computation: RNA solutions to chess problems. *Proc. Natl. Acad. Sci. USA*, 97:1385 – 1389, 2000.
8. L. Feng, S. H. Park, J. H. Reif, and H. Yan. A two-state DNA lattice switched by DNA nanoactuator. *Angew. Chem. Int. Ed.*, 42:4342–4346, 2003.
9. A. A. Henry and F. E. Romesberg. Beyond A, C, G, and T: augmenting nature's alphabet. *Curr. Opin. Chem. Biol.*, 7:727–733, 2003.
10. T. H. LaBean, H. Yan, J. Kopatsch, F. Liu, E. Winfree, J. H. Reif, and N. C. Seeman. The construction, analysis, ligation and self-assembly of DNA triple crossover complexes. *J. Am. Chem. Soc.*, 122:1848–1860, 2000.
11. L. F. Landweber, R. J. Lipton, and M. O. Rabin. DNA^2 DNA computations: A potential 'Killer App'? In H. Rubin and D. H. Wood, editors, *DNA Based Computers III: DIMACS Workshop, June 23-27, 1997, University of Pennsylvania*, pages 161–172, Providence, Rhode Island, 1997. American Mathematical Society.
12. J. Li and W. Tan. A single DNA molecule nanomotor. *Nano Lett.*, 2:315–318, 2002.
13. R. J. Lipton. DNA solution of hard computational problem. *Science*, 268:542–545, 1995.
14. D. Liu and S. Balasubramanian. A proton fuelled DNA nanomachine. *Angew. Chem. Int. Ed.*, 42:5734–5736, 2003.
15. Q. Liu, L. Wang, A. G. Frutos, A. E. Condon, R. M. Corn, and L. M. Smith. DNA computing on surfaces. *Nature*, 403:175–179, 2000.
16. C. Mao, T. H. LaBean, J. H. Reif, and N. C. Seeman. Logical computation using algorithmic self-assembly of DNA triple-crossover molecules. *Nature*, 407:493–496, 2000.
17. C. Mao, W. Sun, and N. C. Seeman. Designed two-dimensional DNA holliday junction arrays visualized by atomic force microscopy. *J. Am. Chem. Soc.*, 121:5437–5443, 1999.
18. C. Mao, W. Sun, Z. Shen, and N. C. Seeman. A DNA nanomechanical device based on the B-Z transition. *Nature*, 397:144–146, 1999.
19. Q. Ouyang, P. D. Kaplan, S. Liu, and A. Libchaber. DNA solution of the maximal clique problem. *Science*, 278:446–449, 1997.
20. J. H. Reif. Parallel molecular computation: Models and simulations. In *Proceedings: 7th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA'95) Santa Barbara,CA*, pages 213–223, 1995.
21. J. H. Reif. Paradigms for biomolecular computation. In C. S. Calude, J. Casti, and M. J. Dinneen, editors, *First International Conference on Unconventional Models of Computation, Auckland, New Zealand*, pages 72–93. Springer Verlag, 1998.
22. J. H. Reif. Local parallel biomolecular computation. In H. Rubin and D. H. Wood, editors, *DNA-Based Computers 3*, volume 48 of *DIMACS*, pages 217–254. American Mathematical Society, 1999.
23. J. H. Reif. The design of autonomous DNA nanomechanical devices: Walking and rolling DNA. *Lecture Notes in Computer Science*, 2568:22–37, 2003. Published in Natural Computing, DNA8 special issue, Vol. 2, p 439-461, (2003).
24. P. W. K. Rothemund. A DNA and restriction enzyme implementation of Turing machines. In R. J. Lipton and E. B. Baum, editors, *DNA Based Computers: Proceedings of the DIMACS Workshop, April 4, 1995, Princeton University*, volume 27, pages 75 – 119, Providence, Rhode Island, 1996. American Mathematical Society.
25. A. J. Ruben and L. F. Landweber. The past, present and future of molecular computing. *Nature Rev. Mol. Cell Biol.*, 1:69–72, 2000.

26. N. C. Seeman. DNA in a material world. *Nature*, 421:427–431, 2003.
27. W. B. Sherman and N. C. Seeman. A precisely controlled DNA biped walking device. *Nano Lett.*, 4:1203–1207, 2004.
28. F. C. Simmel and B. Yurke. Using DNA to construct and power a nanoactuator. *Phys. Rev. E*, 63:041913, 2001.
29. F. C. Simmel and B. Yurke. A DNA-based molecular device switchable between three distinct mechanical states. *Appl. Phys. Lett.*, 80:883–885, 2002.
30. W. D. Smith. DNA computers in vitro and in vivo. In R. J. Lipton and E. B. Baum, editors, *DNA Based Computers: Proceedings of the DIMACS Workshop, April 4, 1995, Princeton University*, pages 121 – 186, Providence, Rhode Island, 1996. American Mathematical Society.
31. A. J. Turberfield, J. C. Mitchell, B. Yurke, Jr. A. P. Mills, M. I. Blakey, and F. C. Simmel. DNA fuel for free-running nanomachines. *Phys. Rev. Lett.*, 90:118102, 2003.
32. A. M. Turing. On computable numbers, with an application to the Entscheidungs problem. In *Proc. London Math. Society Ser. II*, volume 42 of *2*, pages 230–265, 1936.
33. A. M. Turing. On computable numbers, with an application to the entscheidungsproblem. In *Proc. London Math. Society Ser. II*, volume 43, pages 544–546, 1937.
34. E. Winfree. On the computational power of DNA annealing and ligation. In R. J. Lipton and E. B. Baum, editors, *DNA Based Computers 1*, volume 27 of *DIMACS*, pages 199–221. American Mathematical Society, 1996.
35. E. Winfree, F. Liu, L. A. Wenzler, and N. C. Seeman. Design and self-assembly of two-dimensional DNA crystals. *Nature*, 394:539–544, 1998.
36. S. Wolfram. *A new kind of science*. Wolfram Media, Inc., Champaign, IL, 2002.
37. H. Yan, T. H. LaBean, L. Feng, and J. H. Reif. Directed nucleation assembly of DNA tile complexes for barcode patterned DNA lattices. *Proc. Natl. Acad. Sci. USA*, 100:8103–8108, 2003.
38. H. Yan, S. H. Park, G. Finkelstein, J. H. Reif, and T. H. LaBean. DNA-templated self-assembly of protein arrays and highly conductive nanowires. *Science*, 301:1882–1884, 2003.
39. H. Yan, X. Zhang, Z. Shen, and N. C. Seeman. A robust DNA mechanical device controlled by hybridization topology. *Nature*, 415:62–65, 2002.
40. P. Yin, A. J. Turberfield, and J. H. Reif. Designs of autonomous unidirectional walking DNA devices. In *DNA Based Computers 10*, 2004.
41. P. Yin, A. J. Turberfield, S. Sahu, and J. H. Reif. Design of an autonomous DNA nanomechanical device capable of universal computation and universal translational motion. Technical Report CS-2004-07, Duke University, Computer Science Department, 2004.
42. P. Yin, H. Yan, X. G. Daniell, A. J. Turberfield, and J. H. Reif. A unidirectional DNA walker moving autonomously along a linear track. *Angew. Chem. Int. Ed.*, 2004. In press.
43. B. Yurke, A. J. Turberfield, Jr. A. P. Mills, F. C. Simmel, and J. L. Neumann. A DNA-fuelled molecular machine made of DNA. *Nature*, 406:605–608, 2000.

A Clocked DNA-Based Replicator

Bernard Yurke[1,2] and David Zhang[2]

[1] Bell Laboratories, 600 Mountain Ave., Murray Hill NJ 07974
[2] California Institute of Technology, Pasadena, California 91125
{yurke, dzhang}@dna.caltech.edu

Abstract. A stepped replicator is described that uses the energy of hybridization to pry the product from the template in order to prevent product inhibition of replication. Toehold-mediated strand displacement is used to reset the system to its initial state after a round of replication. It is argued that the formation of dimers between structures in the process of replicating should not be able to form and, consequently, the system should exhibit exponential growth.

1 Introduction

The ability to reproduce is one of the defining characteristics of life. This ability arises from a process by which DNA serves as its own template to make copies of itself [1]. There is considerable interest in the construction of synthetic systems that are also capable of template-directed replication [2]. For our purposes synthetic systems will mean systems that do not take advantage of biologically derived polymerases to achieve replication of nucleic acids. Much of the interest in synthetic replication systems comes from the insights they might provide on how life originated or functions [3, 4]. There is interest also from the point of view of what template replication may have to offer for manufacturing. Replication by living organisms is an exponential growth process. In a manufacturing setting this kind of replication would alow for easy scale-up of the volume of units produced. Another feature exhibited by living organisms is that the replication process is not perfect. This allows for Darwinian evolution. The power of directed evolution has been exploited in research and manufacturing settings [5]. Error-prone synthetic replicators may also allow for the directed evolution of useful products.

A number of synthetic chemical systems have been constructed which exhibit template-directed replication [6] or self-replication in oligonucleotide-based systems [7]-[15], peptide-based systems [16]-[20], and other chemical systems [21]-[27]. A characteristic problem of these systems is that the product remains bound to the template or competes with monomers for binding with the template. Such systems exhibit parabolic rather than exponential growth. The replication process tends to stall as the concentration of product increases. Exponential growth is a prerequisite for selection in the Darwinian sense [28, 29]. The sub-exponential growth exhibited by these systems is thus nonconducive to Darwinian evolution.

Recently a self-replicating system based on a ligase ribozyme has been constructed [30] that, at least, at early times exhibits exponential growth.

Recently exponential growth has also been demonstrated for a system employing a stepwise 'feeding' procedure and immobilization of the product on a support [31]. In this system a template is immobilized on a solid support. Fragments of the product that bind to the template are introduced. The fragments are then ligated together to produce the product. Next, a denaturing step releases the product which is then immobilized on fresh solid support. The problem of the binding of a template with its product is thus overcome by immobilization.

Various DNA-based nanostructures [32]-[38] and nanomachines [39]-[46] have been constructed. For a recent review see [38]. This suggests that DNA may be a suitable medium for the construction of synthetic replicators. Most of the DNA-based nanomachines that have been constructed [40]-[46] to date use the energy of hybridization to activate the machines. In addition, strand displacement through competitive binding mediated by toeholds is generally used to return the machine to its initial state. The operation of these machines involves stepwise feeding and, hence, are clocked in the sense that the operator controls when the machine advances to its next state through the addition of the appropriate DNA strand to the solution containing the machines. Here we argue that clocked replicators can be constructed in which the energy of hybridization is used to pull the template-replicated product from the template to allow for template replication. Toehold-mediated strand displacement is used to clear the spent replication machinery from the template and product to prepare for the next round of template replication. The replication process proceeds in a clocked manner in which the replication process is sequenced through a series of steps via the addition of DNA strands in an appropriate sequence. In the discussion that follows it will be assumed that the DNA strands are added in stoichiometry and that one waits for the reaction at each step to go to completion. Alternatively, strands or components could be added in excess, if a means is provided for removing the unreacted excess before one proceeds to the next step. This could, in principle, be achieved through the use of magnetic beads functionalized with the appropriate base sequences to scavenge the unreacted strands. It is argued that the replication scheme we propose is not stalled by product-template binding and exhibits exponential growth. If the template-directed replication process is made a bit faulty this system could, in principle, be subjected to directed evolution.

2 The Replicator

The components from which the replicating structures are constructed are shown in Figure 1. Key components are a set of structures we will refer to as "bases" labeled B_i where i is an integer index distinguishing members of the set. These structures will play a role very similar to individual bases on a DNA strand. Each base B_i consists of two strands of DNA held together by complementary regions labeled n and $\bar{n}$. We will refer to this as the neck region of the base. For

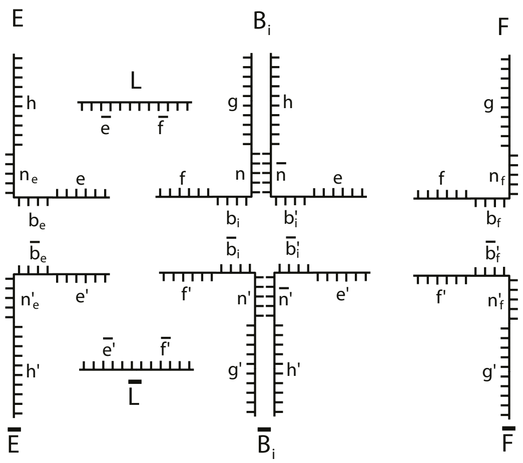


Fig. 1. The components from which the structures to be template-replicated are constructed. The B_i are referred to as bases and the $\bar{B}_i$ as their complements. The linker strands L and $\bar{L}$ allow the bases or the complements, respectively, to polymerize into linear chains. The DNA strands E, $\bar{E}$, F, and $\bar{F}$ serve as chain terminators. The recognition groups b_e, b_f, and the b_i and b'_i allow for binding with the complements via the complementary sequences $\bar{b}_e$, $\bar{b}_f$, and the $\bar{b}_i$ and $\bar{b}'_i$. See the text for further details

each base B_i there is a corresponding structure labeled $\bar{B}_i$ which we will refer to as the complement of the base. This complement also consists of two strands of DNA held together by complementary regions labeled n' and $\bar{n}'$. A base B_i and its complement $\bar{B}_i$ can bind through hybridization of the single-stranded regions b_i and b'_i of the base with the corresponding complementary regions $\bar{b}_i$ and $\bar{b}'_i$ of the complement of the base. In addition to the bases, we have two strands of DNA labeled E and F which can be viewed as a half of a base. Associated with E and F are $\bar{E}$ and $\bar{F}$ that will be referred to as the complements of E and F, respectively. E can bind with $\bar{E}$ via complementary base paring of b_e with $\bar{b}_e$. Similarly, F can bind with $\bar{F}$ via complementary base paring of b_f with $\bar{b}_f$.

The base sequences of the regions labeled e, f, g, and h are the same for all the bases B_i. In addition, the sequences e and h also appear in E. Similarly, the sequences f and g also appear in F. The sequences e', f', g', and h' are likewise shared among $\bar{E}$, $\bar{F}$, and the $\bar{B}_i$. By using identical sequences in portions of these structures one can reduce the size of the set of maximally noninteracting base sequences that one needs to design. The sequences n_e, n'_e, n_f, and n'_f are a set of maximally noninteracting base sequences which are also maximally noninteracting with n, n' and their complements. The base sequences g, g', h, and h' will provide regions for the attachment of the replication machinery, to be discussed in Section 3.

The regions labeled e and f allow the bases B_i to be linked end to end in a polymer chain. E and F provide chain terminations, since L can bind with these strands only at one end. The linkage is performed by the strand labeled L in

Fig. 1 that consists of the concatenation of $\bar{e}$ and $\bar{f}$, the complements of e and f. Similarly, the complements $\bar{B}_i$ can be linked together via the strand labeled $\bar{L}$. For the complements, $\bar{E}$ and $\bar{F}$ serve as chain terminators.

Figure 2 shows the simplest chain that can be constructed for the complements of the bases. It consists of the termination $\bar{E}$, a single base complement $\bar{B}_i$, and the termination $\bar{F}$. These units are held together by the linker strand $\bar{L}$. More generally, one could have an arbitrary sequence of the $\bar{B}_i$ between the two terminations. However, to keep the complexity of the figures and the discussion at a manageable level, we will consider this three-unit system with the understanding that the discussion can be easily extended to the case of a general sequence of $\bar{B}_i$.

The recognition groups $\bar{b}_e$, $\bar{b}_f$, $\bar{b}_i$, and $\bar{b}_i'$ allow template-guided ordering of E, F, and the B_i when they are added to a solution containing the construct of Fig. 2. This is shown in Fig. 3. Because E and F are not connected, the single-stranded pairs b_e with $\bar{b}_e$, b_i with $\bar{b}_i$, b_i' with $\bar{b}_i'$, and b_f with $\bar{b}_f$ are able to wrap around each other to form a double helix.

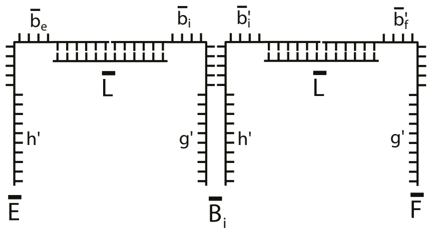


Fig. 2. A trimer consisting of the units $\bar{E}$, $\bar{B}_i$, and $\bar{F}$ linked together by the strand $\bar{L}$. The single-stranded regions $\bar{b}_e$, $\bar{b}_i$, $\bar{b}_i'$, and $\bar{b}_f$ sever as recognition sites for the template-directed construction of the complementary trimer

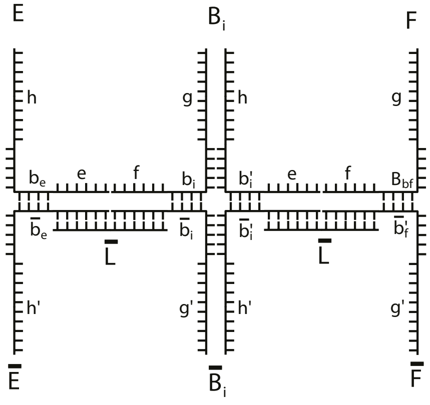


Fig. 3. The construct, consisting of $\bar{E}$, $\bar{B}_i$, and $\bar{F}$ linked together by $\bar{L}$ strands, allows the template-guided arrangement of E, B_i, and F through the pairing of complementary regions b_e with $\bar{b}_e$, b_i with $\bar{b}_i$, b_i' with $\bar{b}_i'$, and b_f with $\bar{b}_f$

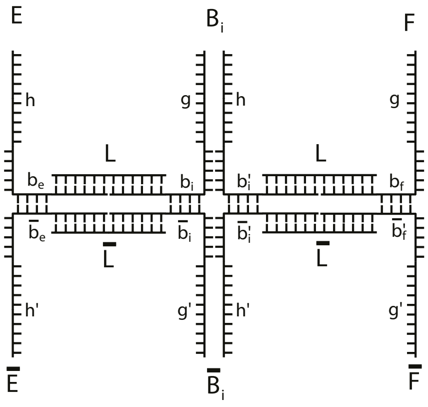


Fig. 4. Addition of L strands to the solution forms the template-constructed polymer consisting of the three units E, B_i, and F. This is the structure that is to be repeatedly template-replicated

It should be noted that the base sequences n and $\bar{n}$ are noncomplementary to the base sequences n' and $\bar{n}'$, and the base sequences b_i and $\bar{b}_i$ are noncomplementary to the base sequences b_i' and $\bar{b}_i'$. As a consequence, the Holiday junction formed when B_i binds to $\bar{B}_i$, as depicted in Figs. 3 and 4, is stable.

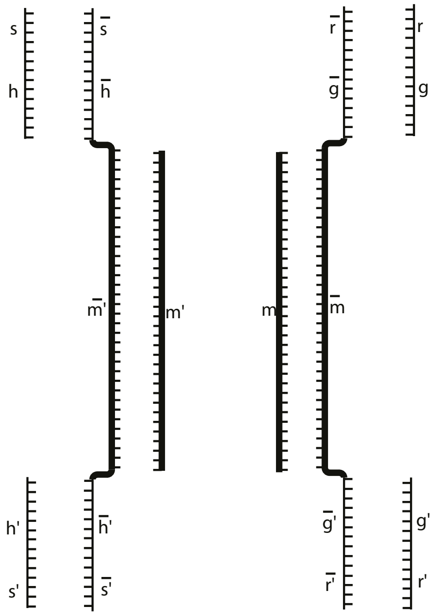


Fig. 5. The DNA strands that constitute the replication machinery. See the text for an explication of the function of these strands. The base sequences m, m', $\bar{m}$, and $\bar{m}'$ involved in pulling apart the complementary polymers are indicated by thicker lines for emphasis

As shown in Fig. 5, the strands L can now be added to the solution to link the e and f sequences of neighboring units. Because there is a nick between the e and f strands attached to a given L and because one has a pivot where b_e is joined with e and where f is joined with b_i, for example, there should be sufficient freedom for L to form a double helix with the e and f strands. The repeated template replication of this structure will now be discussed.

3 The Functioning of the Template-Replication Machinery

Once the structure of Fig. 4 has been formed, in order to have further template replication the two polymers EB_iF and $\bar{E}\bar{B}_i\bar{F}$ must be pulled apart to expose the sequences b_e, b_i, b_i', and b_f and the complements $\bar{b}_e$, $\bar{b}_i$, $\bar{b}_i'$, and $\bar{b}_f$. The template-replication procedure described here uses the DNA strands shown in Fig. 5. The strands consisting of the sequences $\bar{r}\bar{g}\bar{m}\bar{g}'\bar{r}'$ and $\bar{s}\bar{h}\bar{m}'\bar{h}'\bar{s}'$ are able to bind with the structure of Fig. 4 through the base sequences $\bar{g}$, $\bar{g}'$, $\bar{h}$, and $\bar{h}'$. The strand m and m' can hybridize with $\bar{r}\bar{g}\bar{m}\bar{g}'\bar{r}'$ and $\bar{s}\bar{h}\bar{m}'\bar{h}'\bar{s}'$, respectively, and will supply the energy to pry apart the two complementary polymers. The strands rg, $g'r'$, sh, and $h's'$ will allow the removal of the replication machinery, once a round of replication has taken place.

Figure 6. shows the $\bar{r}\bar{g}\bar{m}\bar{g}'\bar{r}'$ and $\bar{s}\bar{h}\bar{m}'\bar{h}'\bar{s}'$ attached to the structure to be replicated. Since the g, g', h, and h' regions are attached to the structure at only one end, it is clear that these are not inhibited from forming double-stranded DNA with complementary regions of $\bar{r}\bar{g}\bar{m}\bar{g}'\bar{r}'$ and $\bar{s}\bar{h}\bar{m}'\bar{h}'\bar{s}'$. To insure that the $\bar{r}\bar{g}\bar{m}\bar{g}'\bar{r}'$ and $\bar{s}\bar{h}\bar{m}'\bar{h}'\bar{s}'$ strands only connect between complementary units rather

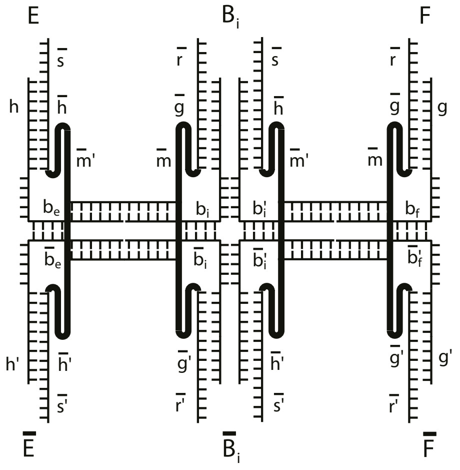


Fig. 6. Upon addition to the solution, the $\bar{r}\bar{g}\bar{m}\bar{g}'\bar{r}'$ and $\bar{s}\bar{h}\bar{m}'\bar{h}'\bar{s}'$ strands attach to the construct of Fig. 4 as shown. For clarity, tick marks indicating the individual bases of the sequences $\bar{m}$ and $\bar{m}'$ have been suppressed

than across units, such as from B_i to $\bar{F}$, L and $\bar{L}$ and the sequences e and f that it hybridizes with should be sufficiently long to prevent the $\bar{r}\bar{g}\bar{m}\bar{g}'\bar{r}'$ and $\bar{s}\bar{h}\bar{m}'\bar{h}'\bar{s}'$ strands from reaching between complementary regions on neighboring units. We will assume that the structure is sufficiently linear so that folding will not be a problem. How long a polymer is allowed is determined by the persistence length of the polymer.

Upon the addition of m and m' to solution they hybridize with the complementary regions in the strands $\bar{r}\bar{g}\bar{m}\bar{g}'\bar{r}'$ and $\bar{s}\bar{h}\bar{m}'\bar{h}'\bar{s}'$. As the double helix is formed $\bar{m}$ and $\bar{m}'$ begin to straighten and pull the structure into two complementary polymers. Since for each base torn apart between, for example, b_e and $\bar{b}_e$, one is formed between, in this case, m' and $\bar{s}\bar{h}\bar{m}'\bar{h}'\bar{s}'$, one has a random walk process that eventually breaks apart the complementary pairs of sequences $(b_e, \bar{b}_e)$, $(b_i, \bar{b}_i)$, $(b_i', \bar{b}_i')$, and $(b_f, \bar{b}_f)$. Since double helices are being unwound during this process, the unit consisting of E and $\bar{E}$ must rotate relative to the unit consisting of B_i and $\bar{B}_i$. That it can do so is evident from the fact that it is connected to the $B_i\bar{B}_i$ unit through the single duplex strand consisting of b_e and $\bar{b}_e$ In fact, the link between the $E\bar{E}$ unit and the $B_i\bar{B}_i$ becomes a single duplex strand at two places, the one already mentioned and the strand consisting of b_i and $\bar{b}_i$. The unit consisting of F and $\bar{F}$ is similarly free to rotate relative to the $B_i\bar{B}_i$ unit. This argument is easily extended to polymers with multiple $B_i\bar{B}_i$

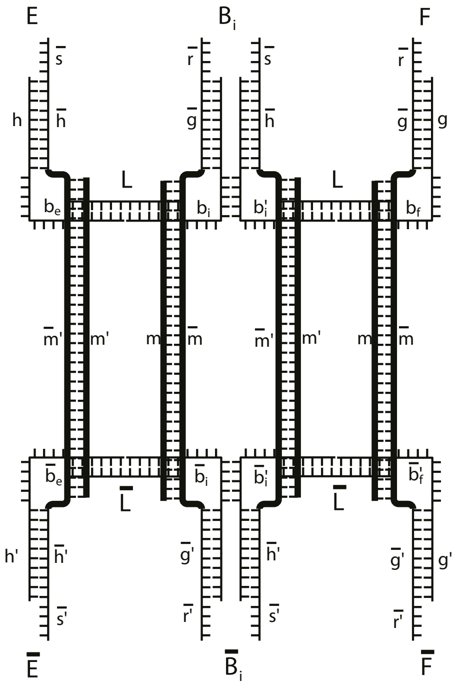


Fig. 7. The construct of Fig. 6 after the motor strands m and m' have been added and the two complementary polymers have been pulled apart

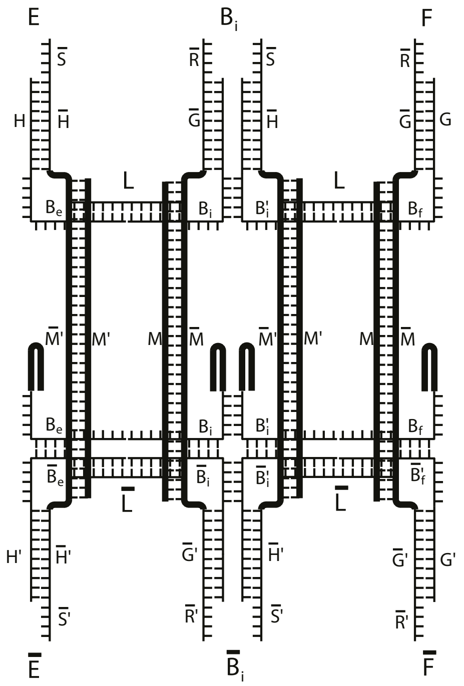


Fig. 8. The construct of Fig. 7 after the addition of the monomers E, B_i, and F. The polymer unit $\bar{E}\bar{B}_i\bar{F}$ serves as a template to put the E, B_i, and F into the correct order. The g and h regions of the monomers have been drawn folded back on themselves with ticks representing individual bases suppressed. Thick lines have been used to represent these regions. This representation will reduce the amount of clutter in Figs. 9 through 11

units. However, the number of turns the end units have to make relative to each other grows linearly with the number of units.

Once the two complementary polymers have been pulled apart, the monomers E, B_i, and F can be added to solution and allowed to hybridize with complementary base sequences on the $\bar{E}\bar{B}_i\bar{F}$ polymer, as shown in Fig. 8. This process takes place in much the same way as discussed in connection with Fig. 3. Again, because there is a free end, the double helices are not inhibited from forming.

It may be of concern that the complementary polymers, when pulled apart, have a propensity to form dimers in which two neighboring constructs, like the one depicted in Fig. 7, bind together, the exposed b_e, b_i, b'_i b_f sites of one construct binding with the complements of another. It is noted that this process is geometrically hindered. In the case when the template and product are pulled apart, each pair of units was free to rotate relative to neighboring pairs by virtue of the fact that in the link between neighboring units are the single duplex strands that need to be unwound. In the present situation this is not the case, because each unit is linked to its neighbors to form a ladder-like structure. The rungs of the ladder are the $\bar{r}\bar{g}\bar{m}\bar{g}'\bar{r}'$ and $\bar{s}\bar{h}\bar{m}'\bar{h}'\bar{s}'$ strands hybridized, re-

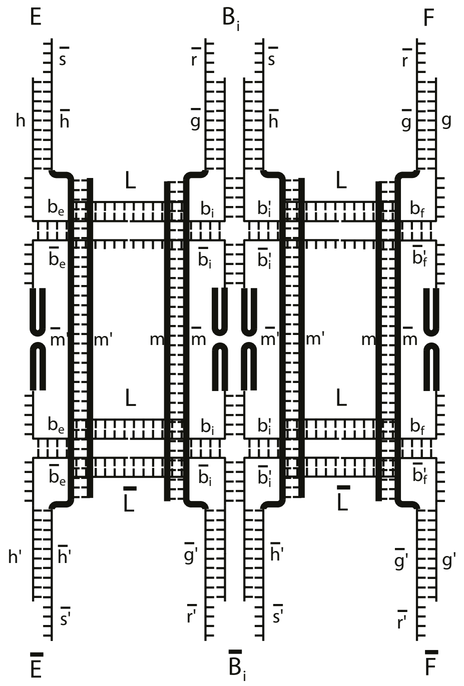


Fig. 9. The construct of Fig. 8 after the addition of the L strands. Now the monomers E, B_i, and F attached to $\bar{E}\bar{B}_i\bar{F}$ are assembled into the linear polymer EB_iF

spectively, with the m and m' strands. The rails of the ladder are the template and its product. Also, apart from the single-stranded recognition regions, the rungs and rails of the ladder are made of stiff double-stranded DNA. Two such ladders forming a dimer do not provide the freedom for a monomer unit and its complement to rotate relative to neighboring pairs of units in order to form duplex DNA. Hence, the duplexes will be incompletely formed. Under such circumstances, monomers will be able to attach themselves to the portions of the recognition regions that remain single-stranded. Then, by competitive binding the monomers will be able to break the duplex bonds. Hence, this system should not exhibit the product inhibition of replication that occurs in most synthetic systems so far devised.

Next the monomers E, B_i, and F are linked together using the L strands. This is shown in Fig. 9. The mechanics of this process is essentially the same as that discussed in connection with figure 4.

Once the new EB_iF polymer has been completed, one adds the monomers $\bar{E}$, $\bar{B}_i$ and $\bar{F}$ to the solution. The recognition sequences of these monomers hybridize with complementary exposed sequences of the upper EB_iF polymer shown in Fig. 9. With the addition of $\bar{L}$ strands a new $\bar{E}\bar{B}_i\bar{F}$ polymer is formed as shown in Fig. 10. Now to produce two copies of the construct of Fig. 4 it is only necessary to remove the hybridization machinery strands.

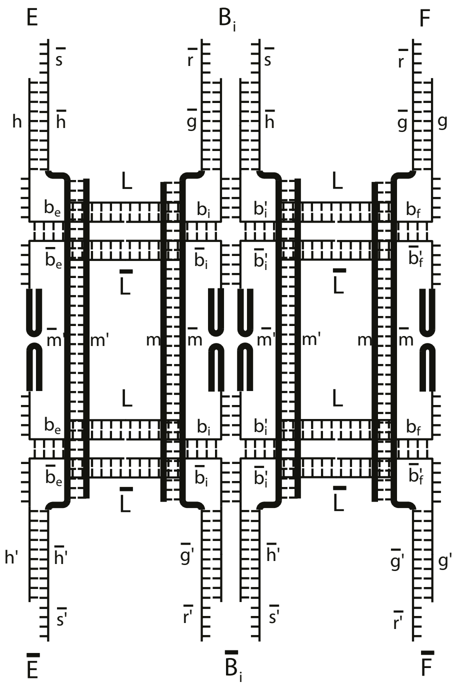


Fig. 10. The construct of Fig. 9 after the addition of monomers $\bar{E}$, $\bar{B}_i$, and $\bar{F}$, followed by the addition of $\bar{L}$. Note that one now has two copies of the construct of Fig. 4, but linked together with the replication machinery strands

The hybridization machinery strands are removed through the addition of the sh, $h's'$, rg, and $g'r'$ strands of Fig. 5. These attach themselves to the $\bar{r}\bar{g}\bar{m}\bar{g}'\bar{r}'$ and $\bar{s}\bar{h}\bar{m}'\bar{h}'\bar{s}'$ strands via the toeholds provided by the base sequences $\bar{r}$, $\bar{r}'$, $\bar{s}$, and $\bar{s}'$. Then, by three-way branch migration, the $\bar{r}\bar{g}\bar{m}\bar{g}'\bar{r}'$ and $\bar{s}\bar{h}\bar{m}'\bar{h}'\bar{s}'$ strands along with the m and m' strands are displaced. As depicted in Fig. 11 the result is two released constructs of the type depicted in Fig. 4 for each initial construct. In addition, the freed hybridization machinery strands are double-stranded over their entire length and, hence, neutral to the addition of further DNA strands to repeat the replication process.

4 Conclusions

We have described a clocked DNA-based template-replication system. It was argued that product inhibition of template formation should not be a problem for this system because geometrical constraints imposed by the attached replication machinery prevent dimer formation between templates and products at the stage when new monomers are added to the system. The system should, thus, exhibit exponential amplification. The system could be made error-prone by choosing the

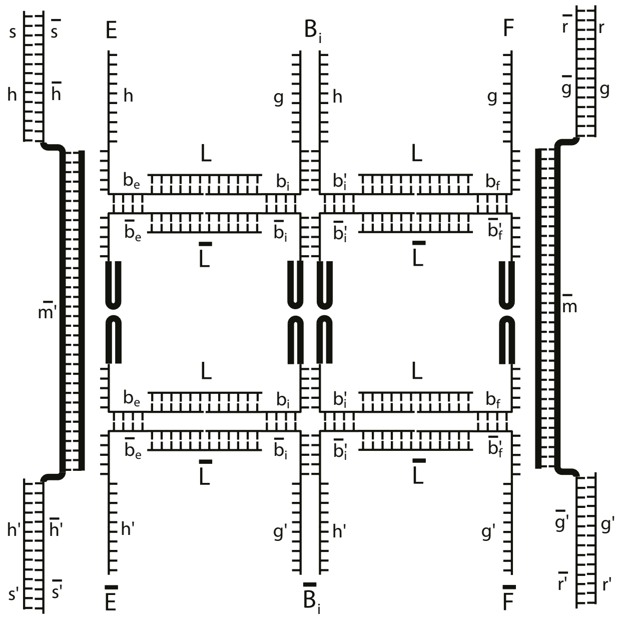


Fig. 11. The addition of strands sh, $h's'$, rg, and $g'r'$ to the construct of Fig. 10 results in the displacement of the hybridization strands from the polymers through competitive binding. The result is the release of two constructs of the same structure as the starting construct, depicted in Fig. 4. Although the hybridization machinery strands are drawn with short displaced segments, these should be viewed as nicks so that, in fact, these strands are double-stranded over their entire length

recognition sequences b_i and b_i' such that the bases B_i are capable of binding to an inappropriate site. In principle, circumstances could be devised where strands with different B_i sequences could compete with each other in a Darwinian fashion where both mutation and selection take place. It is also worth noting that the ladder-like structure formed by the replication machinery as the template and its product are pulled apart can be viewed as a kind of compartmentalization which prevents interference, in this case dimer formation and product inhibition, by neighboring replicators. Also, note that as in biological systems where, during replication, two duplex strands are made from one, here also duplex structures are being duplicated.

References

1. B. Alberts, "DNA replication and recombination," Nature **421**, 431-435 (2003).
2. E. A. Wintner, M. M. Conn, and J. Rebek, "Studies in molecular replication," Acc. Chen. Res. **27**, 198-203 (1994).
3. L. S. Penrose, "Self-Reproducing Machines," Scientific American **200**(6), 105-114 (1959).

4. L. E. Orgel, "Unnatural Selection in Chemical Systems," Acc. Chem. Res. **28**, 109-118 (1995).
5. J. J. Bull and H. A. Wichman, "Applied Evolution," Annu. Rev. Ecol. Syst. **32**, 183-271 (2001).
6. W. K.Johnston, P. J. Unrau, M. S. Lawrence, M. E. Glasner, and D. P. Bartel, "RNA-Catalyzed RNA Polymerization: Accurate and General RNA-Templated Primer Extension," Science **292**, 1319-1325 (2001).
7. G. von Kiedrowski, "A self-replicating hexadeoxynucleotide," Angew. Chem. Int. Edn Engl. **25**, 932-935 (1986).
8. W. S. Zielinski and L. E. Orgel, "Autocatalytic synthesis of a tetranucleotide analogue," Nature **327**, 346-347 (1987).
9. G. von Kiedrowski, B. Wlotzka, J. Helbing, M. Matzen, and S. Jordan, "Parabolic growth of a hexadeoxynucleotide analogue bearing a 3'-5'-phosphoamidate link," Angew. Chem. Int. Edn Engl. **30**, 423-426, 892 (1991).
10. T. Achilles and G. von Kiedrowski, "A self-replicating system from three precursors," Angew. Chem. Int. Edn **32**, 1198-1201 (1993).
11. D. Sievers and G. von Kiedrowski, "Self-replication of complementary nucleotide-based oligomers," Nature **369**, 221-224 (1994).
12. T. Li and K. C. Nicolaou, "Chemical self-replication of palindromic duplex DNA," Nature **369**, 218 (1994).
13. B. Martin, R. Micura, S. Pitsch, and A. Eschenmoser, "Pyranosyl-RNA: further observations on replication," Helv. Chim. Acta **80**, 1901-1951 (1997).
14. D. Sievers and G. von Kiedrowski, "Self-replication of hexadeoxynucleotide analogues: autocatalysis versus cross-catalysis," Chem. Eur. J. **4**, 629-641 (1998).
15. H. Schöneborn, J. Bülle, and G. von Kiedrowski, "Kinetic monitoring of self-replicating systems through measurement of fluorescence resonance energy transfer," Chembiochem **2**, 922-927 (2001).
16. D. H. Lee, J. R. Granja, J. A. Martinez, K. Severin, and M. R. Ghadiri, "A self-replicating peptide," Nature **382**, 525-528 (1996).
17. K. S. Severin, D. H. Lee, J. A. Martinez, and M. R. Ghadiri, "Peptide self-replication via template-directed ligation," Chem. Eur. J. **3**, 1017-1024 (1997).
18. K. Severin, D. H. Lee, J. A. Martinez, M. Vieth, and M. R. Ghadiri, "Dynamic error correction in autocatalytic peptide networks," Angew, Chem. Int. Edn Engl. **37**, 126-128 (1998).
19. S. Yao, I. Ghosh, R Zutshi, and J. A. Chmielewski, "A self-replicating peptide under ionic control," Angew. Chem. Int. Edn Engl. **37**, 478-481 (1998).
20. A. Saghathelian, Y. Yokobayashi, K. Soltani and M. R. Ghadiri, "A chiroselective peptide replicator," Nature **409**, 797-801 (2001).
21. T. Tjivikua, P. Ballester, and J. A. Rebek, "A self-replicating system," J. Am. Chem. Soc. **112**, 1249-1250 (1990).
22. A. Terfort and G. von Kiedrowski, "Self-replication during condensation of 3-aminobenzamidines with 2-formylphenoxyacetic acids," Angew. Chem. Int. Edn Engl. **31**, 654-656 (1992).
23. J.-I. Hong, Q. Fang, V. Rotello, and J. Rebek, "Competition, cooperation, and mutation improving a synthetic replicator by light irradiation," Science **255**, 848-850 (1992).
24. Q. Fang, T. K. Park, and J. Rebek, "Crossover reactions between synthetic replicators yield active and inactive recombinants," Science **256**, 1179-1180 (1992).
25. R. J. Pieters, I. Huc, and J. Rebek, "Reciprocal template effect in a replication cycle," Angew. Chem. Int. Edn Engl. **106**, 1579-1581 (1994).

26. D. N. Reinhoudt, D. M. Rudkevich, and F. de Jong, "Kinetic analysis of the Rebek self-replicating system: Is there a controversy?" J. Am. Chem. Soc. **118**, 6880-6889 (1996).
27. B. Wang and I. O. Sutherland, "Self-replication in a Diels-Alder reaction," Chem. Commun. **16**, 1495-1496 (1997).
28. E. Szathmáry and I. Gladkih, "Sub-exponential growth and coexistence of non-enzymatically replicating templates," J. Theor. Biol. **138**, 55-58 (1989).
29. P. R. Wills, S. A. Kauffman, and B. M. R. Stadler, "Selection dynamics in autocatalytic systems: Templates replicating through binary ligation," Bull. Math. Bio. **60**, 1073-1098 (1998).
30. N. Paul and G. F. Joyce, "A self-replicating ligase ribozyme," PNAS **99**, 12733-12740 (2002).
31. A. Luther, R. Brandsch, and G. von Kiedrowski, "Surface-promoted replication and exponential amplification of DNA analogues," Nature **396**, 245-248 (1998).
32. J. Chen and N. C. Seeman, "The synthesis from DNA of a molecule with the connectivity of a cube," Nature **350**, 631-633 (1991).
33. N. C. Seeman, "Nucleic acid nanostructures and topology," Angew. Chem. Int. Edn Engl. **37**, 3220-3238 (1998).
34. E. Winfree, F. Liu, L. A. Wenzler, N. C. Seeman, "Design and self-assembly of two-dimensional DNA crystals," Nature **394**, 539-544 (1998).
35. C. Mao, W. Sun, N. C. Seeman, "Designed two-dimensional DNA Holiday junction arrays visualized by atomic force microscopy" J. Am. Chem. Soc. **121**, 5437-5443 (1999).
36. T. H. LaBean, H. Yan, J. Kopatsch, F. Liu, E. Winfree, J. H. Reif, N. C. Seeman, "Construction, analysis, ligation, and self-assembly of DNA triple crossover complexes," J. Am. Chem. Soc. **122**, 1848-1860 (2000).
37. C. Mao, T. H. LaBean, J. H. Reif, and N. C. Seeman, "Logical computation using algorithmic self-assembly of DNA triple-crossover molecules," Nature **407**, 493-496 (2000).
38. N. C. Seeman, "DNA in a material world," Nature **421**, 427-431 (2003).
39. C. Mao, W. Sun, Z. Shen, and N. C. Seeman, "A nanomechanical device based on the B-Z transition of DNA," Nature **297**, 144-146 (1999).
40. B. Yurke, A. J. Turberfield, A. P. Mills, Jr., F. C. Simmel, and J. L. Neumann, "A DNA-fuelled molecular machine made of DNA," Nature **406**, 605-608 (2000).
41. F. C. Simmel and B. Yurke, "Using DNA to construct and power a nanoactuator," Phys. Rev. E **63**, art. no. 041913 (2001).
42. F. C. Simmel and B. Yurke, "A DNA-based molecular device switchable between three distinct mechanical states," Appl. Phys. Lett. **80**, 883-885 (2002).
43. H. Yan, X. Zhang, Z. Shen, and N. C. Seeman, "A robust DNA mechanical device controlled by hybridization topology," Nature **415**, 62-65 (2002).
44. J. J. Li and W. Tan, "A single DNA molecular nanomotor," Nano Lett. **2**, 315-318 (2002).
45. P. Alberti and J-L Mergny "DNA duplex-quadruples exchange as the basis for a nanomolecular machine," PNAS **100**, 1569-1573 (2003).
46. L. Feng, S. H. Park, J. H. Reif, H. Yan, "A two-state DNA lattice switched by DNA nanoactuator," Angew. Chem. Int. Ed. **42**, 4342-4346 (2003).

A Bayesian Algorithm for In Vitro Molecular Evolution of Pattern Classifiers

Byoung-Tak Zhang[1,2] and Ha-Young Jang[1]

[1] Biointelligence Laboratory,
Seoul National University, Seoul 151-742, Korea
[2] Computer Science and Artificial Intelligence Laboratory (CSAIL),
MIT, Cambridge, MA 02139
{btzhang, hyjang}@bi.snu.ac.kr
http://bi.snu.ac.kr/

Abstract. We use molecular computation to solve pattern classification problems. DNA molecules encode data items and the DNA library represents the empirical probability distribution of data. Molecular bio-lab operations are used to compute conditional probabilities that decide the class label. This probabilistic computational model distinguishes itself from the conventional DNA computing models in that the entire molecular population constitutes the solution to the problem as an ensemble. One important issue in this approach is how to automatically learn the probability distribution of the DNA-based classifier from observed data. Here we develop a molecular evolutionary algorithm inspired by directed evolution, and derive its molecular learning rule from Bayesian decision theory. We investigate through simulation the convergence behaviors of the molecular Bayesian evolutionary algorithm on a concrete problem from statistical pattern classification.

1 Introduction

Pattern classification is a classical and fundamental problem in artificial intelligence and machine learning with a wide range of applications including computer vision, data mining, information retrieval, and bioinformatics. The task of a pattern classifier is to assign a class to an input pattern. A variety of pattern classification techniques have been developed so far (see, for example, [4]).

This paper explores the potential of molecular computing to solve the computational problems involved with pattern classification. We focus on the probabilistic formulation of the pattern classification problem. The objective is to build a joint probability $P(X, Y)$ of input pattern X and output class Y. Once this model is constructed, class decisions can be made by computing the conditional probabilities, such as $P(Y|X)$. In doing so, we make use of DNA-based molecular computation.

We use a library (as test tube or in some other format) of DNA molecules to represent the probability distribution of data. Each molecule encodes an instance of training data and the frequency of molecules is proportional to the probability of observing the patterns stored in the library. This makes the whole library of molecules a probabilistic pattern classification device.

In this paper we develop a molecular algorithm for learning probabilistic pattern classifiers. It is motivated by in vitro evolution [15] and we derive from Bayesian decision theory a rule for setting the learning parameters for evolving the classifier using a probabilistic DNA-library.

The paper is organized as follows. In Section 2 we describe the probabilistic approach to pattern classification and provide the probability-theoretical basis of using DNA molecules for solving this problem. Section 3 discusses molecular computation of probability functions related with pattern classification. Section 4 addresses the learning problem and presents a molecular algorithm inspired by in vitro evolution. Section 5 shows the mathematical background of the molecular algorithm by deriving its learning rule. Section 6 presents and discusses simulation results. Section 7 draws conclusions.

2 Pattern Classification and DNA Molecules

The aim is to build a pattern classification system f that outputs a label y given an input pattern $\mathbf{x} = (x_1, ..., x_n)$, i.e. $f(\mathbf{x}) = y$. In this paper we restrict ourselves to the case of binary variables. It is convenient to assume there exists a (unknown) target system f^* as an ideal model for f. We also assume f^* behaves according to the probability distribution $P^*(Y|\mathbf{x})$, but the exact form of probability model is unknown. The only information we have is data collected from the input-output pairs of f^*.

DNA computing provides a promising approach to realizing the pattern classifier. We can represent each input pattern as a sequence of A, T, G, and C. The output label can also be encoded as a DNA sequence. For example, if we use 10-mer to encode each binary variable and if there are 30 variables for input pattern and one variable for class label, then DNA molecules of length 310-mer can represent an instance of the training example. The training patterns can be stored as a DNA library and given a query pattern $\mathbf{x}_q$, the molecular pattern classifier compares every library element against $\mathbf{x}_q$ to make class-label decisions. The use of DNA molecules as a memory device and the use of biochemical techniques for memory storage and retrieval provides interesting properties, such as massive parallelism, associative search, fault-tolerance, and miniaturization [2, 3, 6].

The method we present here is in some sense an extension of the memory-based learning (MBL) approach [8]. In MBL, the training examples are stored one copy for each instance. Here, we use many copies for each training example. The number of copies of library elements is updated as new training examples are observed so that their frequency is proportional to their probability of observation. MBL does rote-learning and thus very fast in storage, but very slow in recall since classification computation is done from scratch. Here we update the probabilistic library on learning. On recall our method works like a look-up table, but the probabilistic distribution of the data in the library facilitates classification computation. Keeping multiple copies of data items can also contribute to the robustness and fault-tolerance of the molecular computing system [9, 13]

The frequentist interpretation of probability builds the theoretical basis of our probabilistic molecular pattern classification model. The basic event in molecular pattern classification involves DNA-hybridization reactions. If n is the total number of DNA strands in the library and n_A is the number of strands with pattern (or hybridization event) A, then the probability of A is defined as the limit

$$P(A) = \lim_{n \to \infty} \frac{n_A}{n}. \tag{1}$$

In molecular computation, the accuracy and reliability of the probability values are supported by the Avogadro-scale number of molecules for representing the population. This offers a novel way of representing the probability distribution of data.

3 Molecular Computing for Pattern Classification

In the previous section we described how to represent the probability distribution using DNA molecules. Essentially, the DNA library represents the joint probability $P(X, Y)$ of the input pattern X and the output class Y. In this section we discuss how the class label can be computed.

One criterion to determine the class label is the maximum a posterior (MAP) decision rule. Here, the classifier computes the probability of each class conditional on the input pattern $\mathbf{x}$, and decides as output the class whose conditional probability is the highest, i.e.

$$y^* = \arg \max_{Y \in y_0, y_1} P(Y|\mathbf{x}) \tag{2}$$

where y_0 and y_1 are the candidate classes (for simplicity, we deal with here the case of binary classes, however, the method is generalizable to an arbitrary number of classes). Specific techniques differ in the methods how to model the probability distribution $P(Y, \mathbf{x})$ and how efficiently to compute the necessary probability values.

Here we use a molecular computational method for pattern classification using the probabilistic DNA-library. Given a query pattern $\mathbf{x}$ we extract from the library all the molecules that match the query. These molecules will have class labels from which we decide the majority label as the class of the query pattern. A class label is a sequence appended to denote the class to which the pattern belongs. The extraction may involve some mismatches due to the potential for formation of double-stranded DNA duplexes. There are a lot of work going on to design the sequences and codeword sets (see, for example, [11] and references therein). From the machine learning point of view, the small error occurred by DNA mismatches offers the possibility of generalization by allowing unobserved patterns to be classified. The decision-making can still be robust because it is based on the statistics of the huge number of molecular samples.

The molecular algorithm for computing the class labels can be summerized as follows.

- 1. Let the library L represent the current empirical distribution $P(X, Y)$.
- 2. Present an input (query) pattern $\mathbf{x}$.
- 3. Classify $\mathbf{x}$ using L as follows:
  - 3.1 Extract all molecules matching $\mathbf{x}$ into M.
  - 3.2 From M separate the molecules into classes:
    - ∗ Extract the molecules with label $Y = y_0 = 0$ into M^0.
    - ∗ Extract the molecules with label $Y = y_1 = 1$ into M^1.
  - 3.3 Compute $y^* = \arg\max_{Y \in \{0,1\}} |M^Y|/|M|$.

In Step 3.1, note that the count $c(\mathbf{x})$ of $\mathbf{x}$ in M approximates the probability of observing the pattern which is called evidence:

$$c(\mathbf{x})/|L| = |M|/|L| \approx P(\mathbf{x}). \tag{3}$$

Step 3.2 essentially computes the frequencies $c(Y|\mathbf{x})$ of molecules belonging to different classes Y. These are an approximation of the conditional probabilities given the pattern, i.e. a posteriori probabilities:

$$c(Y|\mathbf{x})/|M| = |M^Y|/|M| \approx P(Y|\mathbf{x}). \tag{4}$$

Thus, in effect, the protocol computes the maximum a posteriori (MAP) criterion:

$$y^* = \arg \max_{Y \in \{0,1\}} c(Y|\mathbf{x})/|M| = \arg \max_{Y \in \{0,1\}} c(Y|\mathbf{x}) \approx \arg \max_{Y \in \{0,1\}} P(Y|\mathbf{x}) \tag{5}$$

It is worth noting that for classification purposes only the relative frequency or concentration of the molecular labels are important.

4 Molecular Computing for Pattern Learning

In the previous section we assumed the library represents the proper joint-probability distribution $P(X, Y)$ of patterns X and their class Y. Here we describe how the library is revised from observed data. Our update procedure is motivated from in vitro evolution [15, 12]. In vitro evolution starts with a library of molecules and evaluates their goodness. Then, the fitter ones are selected as the basis for generating mutants that build the next generation of library. The iteration of the selection-amplification cycle can come up with the identification of molecules that best fits to the target function. In vitro evolution has been used to identify active compounds from composite mixtures [7]. In recent years, a number of methods have been developed to isolate molecules with desired functions from libraries of small organic molecules, nucleic acids, proteins, peptides, antibodies or single-chain antibody fragments, or other polymers [5]. In vitro evolution has also been used to design genetic circuits [16], finite state machines [10], and game programs [14].

We start with a random collection of DNA strands. Each DNA sequence represents an instance $(\mathbf{x}, y)$ of a vector (X, Y) of random variables of interest in the problem domain. Without any prior knowledge the DNA sequences are generated to represent uniform distribution of the data variables. As a new training example $(\mathbf{x}, y)$ is observed, we extract from the library the patterns matching $\mathbf{x}$. The class y^* of $\mathbf{x}$ is determined by the classification procedure described in the previous section. Then, the matching patterns are modified in their frequency depending on their contribution to the correct or incorrect classification of $\mathbf{x}$. If the label v of the library pattern $(\mathbf{u}, v)$ matching $\mathbf{x}$ is correct, i.e. $v = y$, it is duplicated: $L \leftarrow L + \{(\mathbf{u}, v)\}$. Optionally, if the label v is incorrect, i.e. $v \neq y$, the matching library pattern is removed from the library: $L \leftarrow L - \{(\mathbf{u}, v)\}$. The update of library in this way more or less like evolutionary computation [1, 17] with the additional feature that the presentation of a training example proceeds one generation of the library (as a population). This is also a learning procedure since the library improves its classification performance as new examples are presented.

The molecular algorithm for the whole evolutionary learning procedure is summarized as follows.

- 1. Let the library L represent the current empirical distribution $P(X, Y)$.
- 2. Get a training example $(\mathbf{x}, y)$.
- 3. Classify $\mathbf{x}$ using L as described in the previous section. Let this class be y^*.
- 4. Update L
  - If $y^* = y$, then $L_n \leftarrow L_{n-1} + \{\Delta c(\mathbf{u}, v)\}$ for $\mathbf{u} = \mathbf{x}$ and $v = y$ for $(\mathbf{u}, v) \in L_{n-1}$,
  - If $y^* \neq y$, then $L_n \leftarrow L_{n-1} - \{\Delta c(\mathbf{u}, v)\}$ for $\mathbf{u} = \mathbf{x}$ and $v \neq y$ for $(\mathbf{u}, v) \in L_{n-1}$.
- 5. Goto step 2 if not terminated.

In Step 4, $\Delta c(\mathbf{u}, v)$ denotes the number of copies of $(\mathbf{u}, v)$. It should be noted that here we make use of the fact that the addition or removal operation can be performed in parallel in DNA computing. Addition operation can be implemented by PCR and removal can be done by extraction of the corresponding molecules. The update process relies upon the reliability of DNA extraction technology. For effective implementation of the learning procedure, experimental protocols should consider advanced techniques for improving extraction efficiencies, such as the refinery or super-extract model. Note also that the learning rule has a parameter Δc that reflects the strength of learning for each training example. How to set this parameter will be addressed in the next section.

5 Derivation of Bayesian Update Rule

What is the theoretical basis of the molecular evolutionary learning algorithm described in the previous section? We consider the evolution of the probability distribution in the library L. Let L_0 denote the initial library and let its probability distribution be $P_0(X, Y)$. As the nth training example $(\mathbf{x}, y)$ is observed,

L_{n-1} is updated to L_n. Thus, the general form of the learning rule is written as $L_n \leftarrow L_{n-1} + \{\Delta c(\mathbf{x}, y)\}$, where all the class-dependent updates are integrated into one term $\{\Delta c(\mathbf{x}, y)\}$. This update entails the revision of the distribution $P_{n-1}(X, Y|\mathbf{x}, y)$ of L_{n-1} into $P_n(X, Y|\mathbf{x}, y)$ of L_n. Thus, in terms of probability the learning rule is rewritten as

$$P_n(X, Y|\mathbf{x}, y) = (1 + \delta)P_{n-1}(X, Y|\mathbf{x}, y), \tag{6}$$

where δ is a learning parameter determining the strength of update.

Using the Bayes rule we can write $P(X, Y|\mathbf{x}, y) = \frac{P(\mathbf{x},y|X,Y)P(X,Y)}{P(\mathbf{x},y)}$. Note that $P(X, Y|\mathbf{x}, y) = P_n(X, Y|\mathbf{x}, y)$ and $P(X, Y) = P_{n-1}(X, Y|\mathbf{x}, y)$. Thus, inserting Eqn. (6) into the Bayes rule we get

$$(1 + \delta)P_{n-1}(X, Y|\mathbf{x}, y) = \frac{P(\mathbf{x}, y|X, Y)P_{n-1}(X, Y|\mathbf{x}, y)}{P(\mathbf{x}, y)}. \tag{7}$$

Therefore,

$$\delta = \frac{P(\mathbf{x}, y|X, Y) - P(\mathbf{x}, y)}{P(\mathbf{x}, y)}. \tag{8}$$

This indicates that the molecular algorithm follows the Bayesian evolutionary update rule [17].

Setting the parameter δ is important to balance the adaptability and stability of the molecular library as a probabilistic model for the data. The larger the value of δ is, the larger gets the change of the distribution. An alternative way to control δ is via the number of copies Δc which dictates how many copies of the current example should be amplified. To see the influence of δ-value on learning effect in terms of Δc, we use the frequentist interpretation to express the probability:

$$P_{n-1}(X, Y|\mathbf{x}, y) \cong c_{n-1}(\mathbf{x}, y)/|L| \qquad P_n(X, Y|\mathbf{x}, y) \cong c_n(\mathbf{x}, y)/|L| \tag{9}$$

where $c_{n-1}(\mathbf{x}, y)$ and $c_n(\mathbf{x}, y)$ are the number of copies of $\mathbf{x}$ before and after the example $(\mathbf{x}, y)$ is presented, respectively. The size $|L|$ of the library is assumed to remain constant. The difference in probability is expressed as

$$\delta P_{n-1}(X, Y|\mathbf{x}, y) = P_n(X, Y|\mathbf{x}, y) - P_{n-1}(X, Y|\mathbf{x}, y) = \frac{c_n(\mathbf{x}, y) - c_{n-1}(\mathbf{x}, y)}{|L|} \tag{10}$$

and from this we have

$$\delta = \frac{\Delta c(\mathbf{x}, y)}{c_{n-1}(\mathbf{x}, y)}. \tag{11}$$

The above equation shows the relationship of the probability amplification factor δ to the number $\Delta c(\mathbf{x}, y)$ of additional copies of molecules. It is interesting that δ is expressed as the amplification ratio of current copies $c_{n-1}(\mathbf{x}, y)$ since this is equivalent to the number of PCR cycles for signal amplification. Thus, the learning parameter δ can be set indirectly by controlling the number of PCR cycles or its fraction.

6 Simulation Results and Discussion

We performed simulations to study the properties of the molecular algorithms before we realize them with wet DNA. The questions we are interested in are:

– Does the evolutionary learning process converge to the best solution available by the training data?
– If yes, how fast is the convergence? And how to control the learning rate?
– To what extent is the molecular algorithm robust against external perturbation?

We use the majority function of input size $n = 13$. This binary function returns 1 if the input pattern contains 6 or more 1s, otherwise it returns 0. The size of the DNA library was 2^{14}. Theoretically, this covers the entire space of examples for this problem (13 inputs plus 1 output). However we generated the library randomly, so the example space is covered only statistically. We simulate this setting considering the case where DNA computation can not cover the whole problem space, which is true in practice.

Initially, an average of K copies of each instance are generated in the library. We experimented on various K values, ranging from 10 to 10^4. Then, learning proceeds by presenting training patterns. We use a training set of N examples and an independent set of N examples for testing the performance of the molecular pattern classifiers. Typically, N ranged from 500 to 1500. It is important to note that though the training set of size N is given, the learning process is on-line, i.e., the probability distribution of the DNA library is updated before the next example is presented.

To answer the questions mentioned above, we ran the experiments by changing the learning-rate parameter δ. We also changed the example presentation sequences. In one set of runs we presented the positive training example and the negative example alternatingly. This is more reasonable procedure since there is the same number of positive examples and negative examples in the majority function. In another set of runs we presented two positive examples and then one negative example alternatingly. This is to test for the robustness of the molecular learning process against some external, statistical perturbation. These sets of experiments were combined with the varying values of δ.

Figure 1 shows the learning behavior of the algorithm. For this simulation experiment, the learning rate was $\delta = 0.1$ and the DNA library maintained 1000 copies of molecules for each training instance. The examples were presented randomly, alternating a positive and a negative example. The training set size was 800, thus this graph shows the learning curve for the first two (random) sweeps through the training set. We observe a monotonic increase in classification rate which was evaluated on a test set of the same size but independent of the training set.

To see the effect of the learning rate on the convergence we ran the same experiment by changing $\delta = 1.0$. The result is shown in Figure 2. It is observed that the classification performance steadily increases until the completion of the

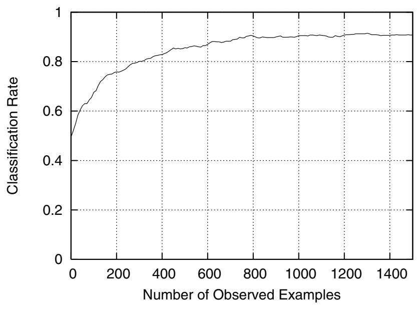


Fig. 1. Learning curve of the simulated DNA-computing classifier: $\delta = 0.1$

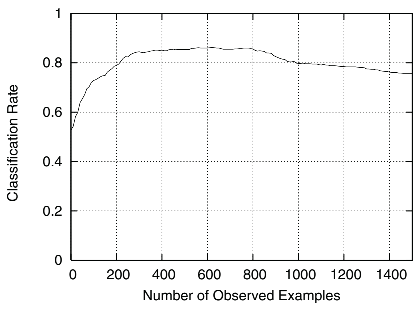


Fig. 2. Learning curve of the simulated DNA-computing classifier: $\delta = 1.0$

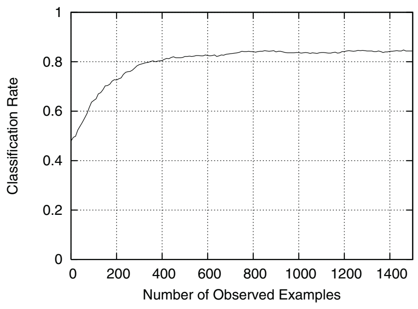


Fig. 3. Learning curve of the simulated DNA-computing classifier: $\delta = 0.1$ with alternating presentations of two positive examples and one negative example

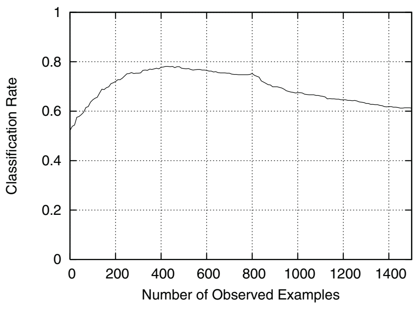


Fig. 4. Learning curve of the simulated DNA-computing classifier: $\delta = 1.0$ with alternating presentations of two positive examples and one negative example

first sweep of the training set, after which the performance degrades. This indicates that the learning rate was set too big. Since we maintained 1000 copies of DNA molecules for each example pattern in this experiment, this means making 1000 copies to learn a single training example may lead to overfitting.

We varied the example presentation order to see the effect of statistically perturbed training sequence. Figures 3 and 4 depict the results which replace the experiments for Figures 1 and 2 by presenting two positive examples and then one negative example alternatingly (and randomly). We observe lower classification performance for these cases than the cases for presenting one positive and one negative example alternatingly. However, when the learning rate is small, the performance still increases steadily, showing robustness against external statistical perturbation. We see some overfitting or instability when the learning rate is too big. For instance, in this run the performance starts to get lower earlier than when the training set correctly reflects the probabilistic distribution of the data space.

To summarize, the simulation results show the learning behavior is relatively stable against the learning rate parameter δ in the range of $0 \leq \delta \leq 1.0$ for the experimental settings we studied. Generally, when δ is set too big, the performance may degrade after a sweep through the training set or later. The system was also relatively robust against the statistical perturbation of example presentation sequences. This seems due to the fact that our learning algorithm has an error-correction component (matched wrong-answering molecules are not copied). Finally, it should also be noted that removing incorrect-answering examples from the library can speed up the learning process, but may lead to instability of the convergence behavior (results not shown). Thus, it should be used with care and only if necessary.

7 Conclusion

We presented a DNA computing algorithm that evolves probabilistic pattern classifiers from training data. Based on Bayesian theory we derived the rule for determining the learning-rate parameter and showed this is related to the number of copies of matched molecules in the DNA library.

We performed simulations to evaluate the performance and stability of the molecular Bayesian evolutionary algorithm. The convergence of the algorithm was quite stable, considering the statistical bias coming from the small number of training examples in our experimental setting. This seems attributed to the probabilistic nature of our molecular computing based on the huge number of molecules to represent the statistical distributions of data. It was also observed that the convergence was relatively stable against external perturbations such as the presentation sequence of training examples.

Our work shows that molecular computation provides several interesting properties for probabilistic computation in general and for pattern classification in specific. The most important property the present work is exploiting, and thus our simulated molecular pattern classifiers are to be useful, is the huge number of molecular-scale data items combined with the highly-parallel molecular recognition mechanism which provides the theoretical and technological basis for the probabilistic DNA library.

Acknowledgements

Thanks are due to Bruce Tidor and the members of the Laboratory for Molecular Science at MIT CSAIL for many fruitful discussions. This research was supported by the National Research Laboratory Program of the Korea Ministry of Science and Technology and by the Next Generation Technology Program of the Korea Ministry of Industry and Commerce.

References

1. Baeck, T., Kok, J.N., Rozenberg, G., "Evolutionary computation as a paradigm for DNA-based computing," *Evolution as Computation*, Landweber, L.F. and Winfree, E. (Eds.), Springer-Verlag, pp. 15-40, 2002.
2. Baum, E. B., "Building an associative memory vastly larger than the brain," *Science*, 268:583-585, 1995.
3. Chen, J. Deaton, R. and Wang, Y.-Z., "A DNA-based memory with in vitro learning and associative recall," *Proc. 9th Annual Meeting on DNA-Based Computers*, pp. 127-136, 2003.
4. Duda, R.O., Hart, P.E., and Stork, D.G., *Pattern Classification*, 2nd Ed. Wiley, 2001.
5. Famulok, M. and Verma, S., "In vivo-applied functional RNAs as tools in proteomics and genomics research," *Trends in Biotechnology*, 20(11):462-466, 2002.
6. Garzon, M. Bobba, K. and Neel, A., "Efficiency and reliability of semantic retrieval in DNA-based memories," *Proc. 9th Annual Meeting on DNA-Based Computers*, pp. 137-149, 2003.
7. Landweber, L.F. and Pokrovskaya, I.D., "Emergence of a dual-catalytic RNA with metal-specific cleavage and ligase activities: The spandrels of RNA evolution," *Proc. Natl. Acad. Sci. USA*, 96:173-178, 1999.
8. Lim, H.-W., Yun, J.-E., Jang, H.-M., Chai, Y.-G., Yoo, S.-I., and Zhang, B.-T., "Version space learning with DNA molecules," *Lecture Notes in Computer Science*, 2568:143-155, 2003.
9. Reif, J. and LaBean, T. "Computationally inspired biotechnology: Improved DNA synthesis and associative search using error-correcting codes and vector quantization," *Lecture Notes in Computer Science*, 2054:145-172, 2001.
10. Rose, J. A., Deaton, R. J., Hagiya, M., Suyama, A., "A DNA-based in vitro genetic program", *Journal of Biological Physics*, 28:493-498, 2002.
11. Shin, S.-Y., Lee, I.-H., Kim, D. and Zhang, B.-T. "Multi-objective evolutionary optimization of DNA sequences for reliable DNA computing," *IEEE Transactions on Evolutionary Computation*, 2004 (to appear).
12. Wilson, D.S. and Szostak, J.W., "In vitro selection of functional nucleic acids," *Ann. Rev. Biochem.*, 68:611-647, 1999.
13. Winfree, E. and Bekbolatov, R. "Proofreading tile sets: Logical error correction for algorithmic self-assembly," Talk at *The 9th Annual Meeting on DNA-Based Computers*, Madison, WI, June 2003.
14. Wood, D. "DNA starts to learn Poker," Proc. DNA7, *Lecture Notes in Computer Science*, 2340:92-103, 2003.
15. Wright, M.C. and Joyce, G.F., "Continuous in vitro evolution of catalytic function," *Science*, 276:614-617, 1997.
16. Yokobayashi, R., Weiss, R., and Arnold, F.H., "Directed evolution of a genetic circuit," *Proc. Natl. Acad. Sci. USA*, 99(26):16587-16591, 2002.
17. Zhang, B.-T., "A unified Bayesian framework for evolutionary learning and optimization," *Advances in Evolutionary Computation*, Chapter 15, pp. 393-412, Springer-Verlag, 2003.

Author Index

Alhazov, Artiom 1

Baryshnikov, Yuliy 14
Bi, Hong 50

Câmpeanu, Cezar 24
Carpenter, Dylan 50
Cavaliere, Matteo 1, 35
Chen, Ho-Lin 62
Chen, Junghuei 50
Coffman, Ed 14
Csuhaj-Varjú, Erzsébet 76

Deaton, Russell 50
Díaz, Carmen Graciani 128
D'yachkov, Arkadii G. 90

Franco, Giuditta 104
Fujibayashi, Kenichi 113

Garzon, Max H. 50, 281
Giagulli, Cinzia 104
Goel, Ashish 62

Hagiya, Masami 202
Harju, Tero 138, 149
Henkel, Christiaan V. 159
Hoogeboom, Hendrik Jan 375

Ibarra, Oscar H. 76
Irie, Satoru 258

Jang, Ha-Young 458
Jonoska, Nataša 35

Kameda, Atsushi 355
Kari, Lila 169, 182
Kawakami, Takashi 258
Keinan, Ehud 35
Kim, Jin Woo 50
Kobayashi, Satoshi 192
Konstantinidis, Stavros 169, 182
Kubota, Mitsuhiro 202

Laudanna, Carlo 104
Lee, Ji Youn 215
Leupold, Peter 224
Li, Chang 138
Lim, Hee-Woong 215

Macula, Antony J. 90
Manca, Vincenzo 104
Margenstern, Maurice 149, 235
Mitrana, Victor 235
Momčilović, Petar 14
Moore, Cristopher 247
Morgan, Clint 247
Murata, Satoshi 113

Ogura, Yusuke 258
Ohuchi, Azuma 355

Păun, Andrei 24
Păun, Gheorghe 268
Park, Tai Hyun 215
Pérez-Jiménez, Mario J. 128, 235
Petre, Ion 138
Phan, Vinhthuy 281
Piran, Ron 35
Pogozelski, Wendy K. 90

Reif, John H. 293, 410, 426
Renz, Thomas E. 90
Rozenberg, Grzegorz 138, 159, 375
Rykov, Vyacheslav V. 90

Sahu, Sudheer 293, 426
Sakamoto, Kensaku 308
Schulman, Rebecca 319
Seelig, Georg 329
Seeman, Nadrian C. 35
Soloveichik, David 344
Someya, Hiroshi 308
Sosík, Petr 169, 182
Spaink, Herman P. 159
Stefanovic, Darko 247
Stojanovic, Milan N. 247
Sumiyama, Fumika 258
Suyama, Akira 258

Takahashi, Naoto 355
Tanida, Jun 258

Torney, David C. 90
Turberfield, Andrew J. 410, 426

van Noort, Danny 365
van Vliet, Rudy 375
Vaszil, György 76

Wakabayashi, Kenichi 389
Wang, Yu-Zhen 50
Winfree, Erik 319, 329, 344
Wood, David 50

Yamamoto, Masahito 355
Yamamura, Masayuki 308, 389
Yan, Hanwen 399
Yin, Peng 293, 410, 426
Yogev, Sivan 35
Yoo, Suk-In 215
Yurke, Bernard 329, 445

Zhang, Byoung-Tak 215, 458
Zhang, David 445