

LNCS 2054

Anne Condon
Grzegorz Rozenberg (Eds.)

DNA Computing

6th International Workshop on DNA-Based Computers, DNA 2000
Leiden, The Netherlands, June 2000
Revised Papers



Springer

Lecture Notes in Computer Science

Edited by G. Goos, J. Hartmanis and J. van Leeuwen

2054

Springer

Berlin

Heidelberg

New York

Barcelona

Hong Kong

London

Milan

Paris

Singapore

Tokyo

Anne Condon Grzegorz Rozenberg (Eds.)

DNA Computing

6th International Workshop on DNA-Based Computers, DNA 2000
Leiden, The Netherlands, June 13-17, 2000
Revised Papers



Springer

Series Editors

Gerhard Goos, Karlsruhe University, Germany
Juris Hartmanis, Cornell University, NY, USA
Jan van Leeuwen, Utrecht University, The Netherlands

Volume Editors

Anne Condon
University of British Columbia
Department of Computer Science
201-2366 Main Mall
Vancouver, B.C., Canada V6T 1Z4
E-mail: condon@cs.ubc.ca
Grzegorz Rozenberg
Leiden University
Leiden Institute of Advanced Computer Science (LIACS)
Niels Bohrweg 1
2333 CA Leiden, The Netherlands
E-mail: rozenberg@liacs.nl

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

DNA computing : revised papers / 6th International Workshop on DNA Based Computers, DNA 2000, Leiden, The Netherlands, June 13 - 17, 2000. Anne Condon ; Grzegorz Rozenberg (ed.). - Berlin ; Heidelberg ; New York ; Barcelona ; Hong Kong ; London ; Milan ; Paris ; Singapore ; Tokyo : Springer, 2001
(Lecture notes in computer science ; 2054)
ISBN 3-540-42076-2

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

ISSN 0302-9743

ISBN 3-540-42076-2 Springer-Verlag Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer-Verlag. Violations are liable for prosecution under the German Copyright Law.

Springer-Verlag Berlin Heidelberg New York
a member of BertelsmannSpringer Science+Business Media GmbH

<http://www.springer.de>

© Springer-Verlag Berlin Heidelberg 2001
Printed in Germany

Typesetting: Camera-ready by author, data conversion by PTP Berlin, Stefan Sossna
Printed on acid-free paper SPIN 10781535 06/3142 5 4 3 2 1 0

Preface

The papers in this volume were presented at the 6th International Meeting on DNA Based Computers, organized by the Leiden Center for Natural Computing and held from June 13 to June 17, 2000 at The Lorentz Center, University of Leiden, Leiden, The Netherlands. DNA Computing is a novel and fascinating development at the interface of computer science and molecular biology. It has emerged in recent years, not simply as an exciting technology for information processing, but also as a catalyst for knowledge transfer between information processing, nanotechnology, and biology. This area of research has the potential to change our understanding of the theory and practice of computing.

The call for papers and poster presentations sought contributions of original research and technical expositions in all areas of bio-computation. A total of 33 abstracts were submitted of which 16 were accepted for presentation and included in the proceedings. The papers were selected by the program committee based on originality and quality of research and on relevance to the bio-computing field. Invited talks were given by Masami Hagiya (Tokyo University), Laura Landweber (Princeton University), John Reif (Duke University), Thomas Schmidt (Leiden University), and Lloyd M. Smith (University of Wisconsin). Invited papers based on the talks by Hagiya and Reif are included in this volume, along with the contributed papers. Additional tutorials were held on the first and last days of the conference.

The conference was held under the auspices of the ACM Special Interest Group on Algorithms and Computation Theory (ACM SIGACT) and the European Association for Theoretical Computer Science (EATCS). We gratefully acknowledge support and sponsorship from the following organizations: the European Molecular Computing Consortium (EMCC), the European Commission (EC) Institute for Programming research and Algorithmics (IPA), the Leiden Institute of Advanced Computer Science (LIACS), the Lorentz Visitor Center (LC), and the Netherlands Organization for Scientific Research (NWO).

The program committee wishes to thank all those who submitted papers for consideration.

March 2001

Anne Condon
Program Chair

Organization

Program Committee

M. Amos	Univ. Liverpool, UK
J. Chen	Univ. Delaware, USA
A. Condon	Univ. British Columbia, Canada (Program Chair)
T. Head	Binghamton Univ., USA
N. Jonoska	Univ. South Florida, USA
L. Landweber	Princeton Univ., USA
G. Paun	Romanian Acad., Romania
G. Rozenberg	Univ. Leiden, The Netherlands
A. Suyama	Univ. Tokyo, Japan
E. Winfree	Caltech, USA
T. Yokomori	Waseda Univ., Japan
B. Yurke	Lucent Technologies, USA

International Organizing Committee

M. Amos	Univ. Liverpool, UK
A. Condon	Univ. British Columbia, Canada
T. Head	Binghamton Univ., USA
L. Kari	Univ. Western Ontario, Canada
G. Rozenberg	Univ. Leiden, The Netherlands (Organizing Committee Chair)
H. Rubin	Univ. Pennsylvania, USA
E. Winfree	Caltech, USA

Table of Contents

Engineered Communications for Microbial Robotics	1
<i>Ron Weiss and Thomas F. Knight, Jr.</i>	
Successive State Transitions with I/O Interface by Molecules	17
<i>Ken Komiya, Kensaku Sakamoto, Hidetaka Gouzu, Shigeyuki Yokoyama, Masanori Arita, Akio Nishikawa, and Masami Hagiya</i>	
Solution of a Satisfiability Problem on a Gel-Based DNA Computer	27
<i>Ravinderjit S. Braich, Cliff Johnson, Paul W.K. Rothmund, Darryl Hwang, Nickolas Chelyapov, and Leonard M. Adleman</i>	
Diophantine Equations and Splicing: A New Demonstration of the Generative Capability of H Systems	43
<i>Pierluigi Frisco</i>	
About Time-Varying Distributed H Systems	53
<i>Maurice Margenstern and Yurii Rogozhin</i>	
String Tile Models for DNA Computing by Self-Assembly	63
<i>Erik Winfree, Tony Eng, and Grzegorz Rozenberg</i>	
From Molecular Computing to Molecular Programming	89
<i>Masami Hagiya</i>	
Graph Replacement Chemistry for DNA Processing	103
<i>John S. McCaskill and Ulrich Niemann</i>	
DNA and Circular Splicing	117
<i>Paola Bonizzoni, Clelia De Felice, Giancarlo Mauri, and Rosalba Zizza</i>	
Molecular Computing with Generalized Homogeneous P-Systems	130
<i>Rudolf Freund and Franziska Freund</i>	
Computationally Inspired Biotechnologies: Improved DNA Synthesis and Associative Search Using Error-Correcting Codes and Vector-Quantization	145
<i>John H. Reif and Thomas H. LaBean</i>	
Challenges and Applications for Self-Assembled DNA Nanostructures	173
<i>John H. Reif, Thomas H. LaBean, and Nadrian C. Seeman</i>	
A Space-Efficient Randomized DNA Algorithm for k -Sat	199
<i>Kevin Chen and Vijay Ramachandran</i>	
A DNA-Based Random Walk Method for Solving k -SAT	209
<i>Sergio Díaz, Juan Luis Esteban, and Mitsunori Ogihara</i>	

Solving Computational Learning Problems of Boolean Formulae on DNA
Computers 220
Yasubumi Sakakibara

The Fidelity of Annealing-Ligation: A Theoretical Analysis 231
John A. Rose and Russell J. Deaton

DNA Implementation of a Royal Road Fitness Evaluation 247
Elizabeth Goode, David Harlan Wood, and Junghuei Chen

Steady Flow Micro-Reactor Module for Pipelined DNA Computations 263
*John S. McCaskill, Robert Penchovsky, Marlies Gohlke,
Jörg Ackermann, and Thomas Rucker*

Author Index 271

Engineered Communications for Microbial Robotics

Ron Weiss and Thomas F. Knight, Jr.*

M.I.T. Artificial Intelligence Laboratory, 545 Technology Square, Cambridge, MA
02139 {rweiss,tk}@ai.mit.edu

Abstract. Multicellular organisms create complex patterned structures from identical, unreliable components. Learning how to engineer such robust behavior is important to both an improved understanding of computer science and to a better understanding of the natural developmental process. Earlier work by our colleagues and ourselves on amorphous computing demonstrates in simulation how one might build complex patterned behavior in this way. This work reports on our first efforts to engineer microbial cells to exhibit this kind of multicellular pattern directed behavior.

We describe a specific natural system, the Lux operon of *Vibrio fischeri*, which exhibits density dependent behavior using a well characterized set of genetic components. We have isolated, sequenced, and used these components to engineer intercellular communication mechanisms between living bacterial cells.

In combination with digitally controlled intracellular genetic circuits, we believe this work allows us to begin the more difficult process of using these communication mechanisms to perform directed engineering of multicellular structures, using techniques such as chemical diffusion dependent behavior. These same techniques form an essential part of our toolkit for engineering with life, and are widely applicable in the field of microbial robotics, with potential applications in medicine, environmental monitoring and control, engineered crop cultivation, and molecular scale fabrication.

1 Introduction

The developmental process requires coordinated, robust action among a very large number of essentially identical, unreliable components. In stark contrast to current computer science engineering practice, these developmental programs are highly fault tolerant. Imagine what would happen if any biological mechanism exhibited the same fragility as a modern microprocessor, operating system, or satellite.

Previous work in our group [1,3,4,24,26,29,30] has looked at some of these robustness and pattern formation issues in simulation, with intriguing results. We

* This work is supported by DARPA/ONR under grant number N00014-96-1-1228 and by NTT Corporation under grant MIT9904-010.

found that the topic we call *amorphous computing* requires a different set of algorithms and a different approach to thinking about structures than conventional computer science.

But we also must better understand the developmental process in a biological context. Although we are making significant progress, we simply do not fully understand the pattern formation of even the simplest of biological structures. But surely concepts from computer science, such as subroutines, divide-and-conquer, recursion and iteration will play a major role in understanding the genetic control of developmental diversity. Both biology and computer science have lessons to learn from a cooperative investigation of this field.

Even simple biological systems can exhibit complex developmental processes. The motile, gram negative bacterium *Myxococcus xanthus*, for example, exhibits social behavior and cellular differentiation during cooperative feeding. The controlled, density dependent, release of antibiotics and cell wall degrading enzymes to kill competitors allows moving swarms (so-called “wolf-packs”) to act more effectively than individuals [8]. Similarly, *M. xanthus* exhibits selection, during starvation, of a small number of cells out of a swarm of 100,000 to change form from rod-like bacteria to environmentally protected spherical myxospores. Spore formation requires high cell density, nutrient limitation, and a solid surface [9, 20].

In this paper we undertake a *biological implementation* of what we believe is a key component in building such developmental pattern engineering techniques – cell to cell communications. Communication between cells is obviously essential to any kind of coordinated expression. But in development, and in our amorphous computing simulations, one kind of communication emerges as especially important – the ability to detect and act on chemical signal concentration gradients. Such gradient dependent expression is the building block of locally unique behavior, as well as the organizing principle which allows the construction of local coordinate systems through the creation and detection of chemical gradients. Such trophic behavior provides one basic organizing principle for complex patterned development.

In this work, we have isolated a specific chemical cell to cell signaling mechanism from a natural biological system, the quorum sensing system of *Vibrio fischeri*. This system encodes genes and promoter sequences which allow the controlled expression of the chemical *Vibrio fischeri* autoinducer (VAI) within one sender cell, and the detection and controlled expression of specific genes in another, receiving cell. The free diffusion of the VAI chemical within the medium and across cell membranes allows the establishment of chemical gradients and the controlled expression of genetic circuits as a result.

Specifically, we demonstrate in this work the construction and testing of engineered genetic circuits which exhibit the ability to send a controlled signal from one cell, diffuse that signal through the intercellular medium, receive that signal within an a second cell, and activate a remote transcriptional response.

In combination with other ongoing work in digitally controlled gene expression [11, 16, 19, 23, 29] this work provides components for a biological substrate for expressing pattern formation. These same components are also a key part of our toolkit for engineering with life, with important implications for medicine, agri-

culture, environmental monitoring, and engineering – including molecular scale manufacturing and molecular electronics.

In the remainder of this paper we describe the mechanism of quorum sensing in bacteria (Sections 2-3), present the plasmids engineered for communications (Section 4), report on our experimental results (Section 5), and offer conclusions and avenues for future work (Section 6).

2 Quorum Sensing in Bacteria

Vibrio fischeri is a gram-negative bioluminescent marine prokaryote which naturally occurs in two distinct environments. In seawater, it swims freely at concentrations of approximately ten cells per liter. It also grows naturally in a symbiotic relationship with a variety of invertebrate and vertebrate sea organisms, especially the Hawaiian sepiolid squid, *Euprymna scolopes* and the Japanese pinecone fish, *Monocentris japonica* [27]. In these symbiotic relationships, the bacteria grow to densities of approximately 10^{10} cells per liter.

In the free living state, *Vibrio fischeri* emits essentially no light (< 0.8 photons/second/cell). In the light organ, however, the same bacteria emit more than 800 photons/second/cell, producing very visible bioluminescence. In culture, *Vibrio fischeri* demonstrates a similar density dependent bioluminescence, with induction occurring at about 10^{10} cells/liter.

Work over many years has established that this behavioral change is due to a natural cell density detection mechanism, which has been termed quorum sensing [15]. The quorum sensing mechanism relies on the synthesis and detection of a very specific, species unique chemical, an *autoinducer*, which mediates intercellular communications. In *Vibrio fischeri*, this autoinducer chemical (VAI) has been identified as N-(3-oxohexanoyl)-3-amino-dihydro-2-(3H)-furanone [10]. The gene, LuxI, catalytic protein, and synthetic pathway for this chemical has also been identified [14].

Briefly, the LuxI gene encodes an acyl-homoserine lactone synthetase which uses highly available metabolic precursors found within most gram negative prokaryotic bacteria – acyl-ACP from the fatty acid metabolic cycle, and S-adenosylmethionine (SAM) from the methionine pathway – to synthesize VAI.

The *Vibrio fischeri* autoinducer (VAI) freely diffuses across the bacterial cell membrane. Thus, at low cell densities, low VAI concentrations are available. Within a light organ, or at high culture densities, VAI builds up within the environment, resulting in a density dependent induction of bioluminescence.

The response mechanism to VAI concentration has also been extensively analyzed [28]. Briefly, the LuxR gene codes for a two domain DNA binding protein which interacts with VAI and the Lux box of the LuxICDABEG operon promoter to exercise transcriptional control. At nanomolar concentrations, VAI binds to the N terminal domain of the LuxR protein, which in turn activates the C-terminal helix-turn-helix DNA binding domain. The LuxR protein acts as a transcriptional activator for the RNA polymerase holoenzyme complex. The activated protein likely binds in dimeric or multimeric forms, because of the evident dyadic symmetry of the Lux box binding domain.

The genetic structure of the *Vibrio fischeri* Lux operon has been established by the successful cloning and expression of the Lux genes into *E. coli* [12]. It is

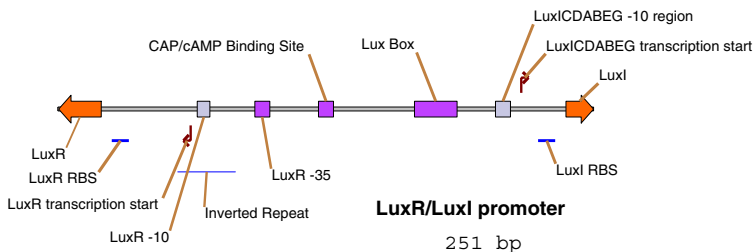


Fig. 1. LuxR and LuxI promoter regions from *Vibrio fischeri*.

somewhat surprising (although common) for the transfer of regulatory genes and entire metabolic pathways to function straightforwardly across gram-negative species boundaries in this way.

Given the potential utility of both the autoinducer control mechanism as a cell to cell signaling mechanism, and the Lux operon as a reporter gene, we undertook to isolate these operons and engineer their interfaces. An initial stumbling block was the lack of complete sequence information. Remarkably, although this system has been the subject of hundreds of papers, a complete sequence of the operons was not available in GENBANK. Therefore, as first step, we undertook to isolate the operon, completely sequence it, and deposit the resulting sequence. That effort is described in Appendix [A](#).

3 Genetic Features of the LuxR/LuxI Operons

The nucleotide structure of the sequenced regulatory region is shown in figure [1](#). This region encodes two divergently transcribed promoters. The left operon constitutively expresses the LuxR transcript, which is coded by the left ORF. This operon has a standard σ^{70} binding region, consisting of a -10 and -35 sequence, and a CRP/CAMP binding site. The CRP/CAMP binding site allows catabolic repression on the left LuxR operon.

The right operon drives expression of the LuxICDABEG transcript, coding for autoinducer production (LuxI) and the bioluminescence cassette of LuxCDABEG. It consists of a standard -10 σ^{70} binding site, but is missing the -35 site. Instead, the *lux box*, a 20 base inverted palindromic repeat, allows dimeric binding of the active form of LuxR binding protein, activating the RNA polymerase holoenzyme complex, under control of the LuxR protein – and hence indirectly, the VAI concentration.

The *lux box* is a common motif in regulatory proteins of the LuxR family, and occurs upstream of many LuxR homologous genes. The sequence of the *lux box* in this construct is 5'(acctgtagga tcgtacaggt); the consensus sequence for similar *lux boxes* in other constructs is [18](#) 5'(rnstgyaxga tnxtrcasrt)3' (n = a, t, g, c; x = n or gap; s = g, c; r = a, g; y = c, t).

Note that the dimeric binding of the LuxR product produces the kind of nonlinear concentration/response behavior discussed in [19,29](#) and widely seen in DNA binding protein transcriptional control. This nonlinear response is an essential element of signal restoration and digital control of expression.

The transcription of the right operon also enhances the production of LuxI, and thus the VAI synthetase, and VAI. We see here the key component of a Schmidt-trigger positive feedback gate – once transcription is turned, the enhancement is self-reinforcing, leading to hysteresis in the transfer curve.

4 Engineered Plasmid Constructs

In order to experiment with intercellular communications, we constructed a series of plasmids, and then transformed them into *E. coli* cells. The plasmids can be roughly categorized into three groups: preliminary plasmids (Section 4.1), plasmids that enable cells to transmit the message by catalyzing the formation of autoinducer (Section 4.2), and plasmids that enable cells to respond to the message through the use of the appropriate region of the lux operon (Section 4.3).

4.1 Preliminary Plasmids

Initially, we constructed a series of plasmids (Figure 2) that could serve as templates for cloning the final sender and receiver plasmids. The first plasmid, pRW7-1, combines the backbone of the general purpose high copy number plasmid pUC19 with GFP(LVA) from Clontech pGFP(LVA). Both pUC19 and pGFP(LVA) were digested with SpeI and XmaI, and the GFP(LVA) CDS and its associated synthetic ribosome binding site (RBSII) were cloned into pUC19. GFP(LVA) is a variant of the green fluorescent protein with a destabilizing tail (amino acids RPAANDENYLVA) that results in a protein half life of approximately 40 minutes.

Next, to produce pRW7-2, a transcription termination region (rrnB T1) based on a sequence from pKK232-8 [25] was cloned into pRW7-1 using two oligonucleotides. The oligos were annealed by incubating @97°C for 10 minutes, then incubating @65°C for 15 minutes, incubating @24°C for 15 minutes, and finally storing @4°C, to produce the following double stranded segment with overhangs that match an AatII and XmaI digest:

```

ACCCGGGAATCCAGGCATCAAATAAACGAAAGGCTCAGTCGAAAGACTGGGCCTTTC GTTTTATCTGTTGTTTGTCGGTGAAACGCTCTCACCGGT
TGCATGGGCCCTTAAGGGTCGTTAGTTTATTTTTCGCTTCCGAGTCAGCTTTCTGACCCGGAAAG CAAATAGACAACAACAGCCACTTCGAGAGTGCCAGGCC

```

The annealed oligos were then ligated into pRW7-2 digested with AatII and XmaI. The plasmid pRW7-3, which includes the same transcription termination region but on the 5' end of GFP(LVA), was constructed in a similar fashion. The oligos used have HindIII and XbaI overhangs, and were cloned into a pRW7-2 HindIII/XbaI digest.

The final preliminary plasmid pRW7-4 includes p(LAC-const), a new constitutive synthetic promoter, in front of the GFP(LVA) CDS. We designed the constitutive promoter p(LAC-const) based on the LAC promoter, as shown in Figure 3. In p(LAC-const), the lacO and CAP binding sites have been removed, and the -10 and -35 regions have been modified to resemble the consensus -10 and -35 regions respectively [22]. p(LAC-const) was introduced into pRW7-3 (digested with AgeI/Acc65I) using a pair of oligos with AgeI and Acc65I overhangs

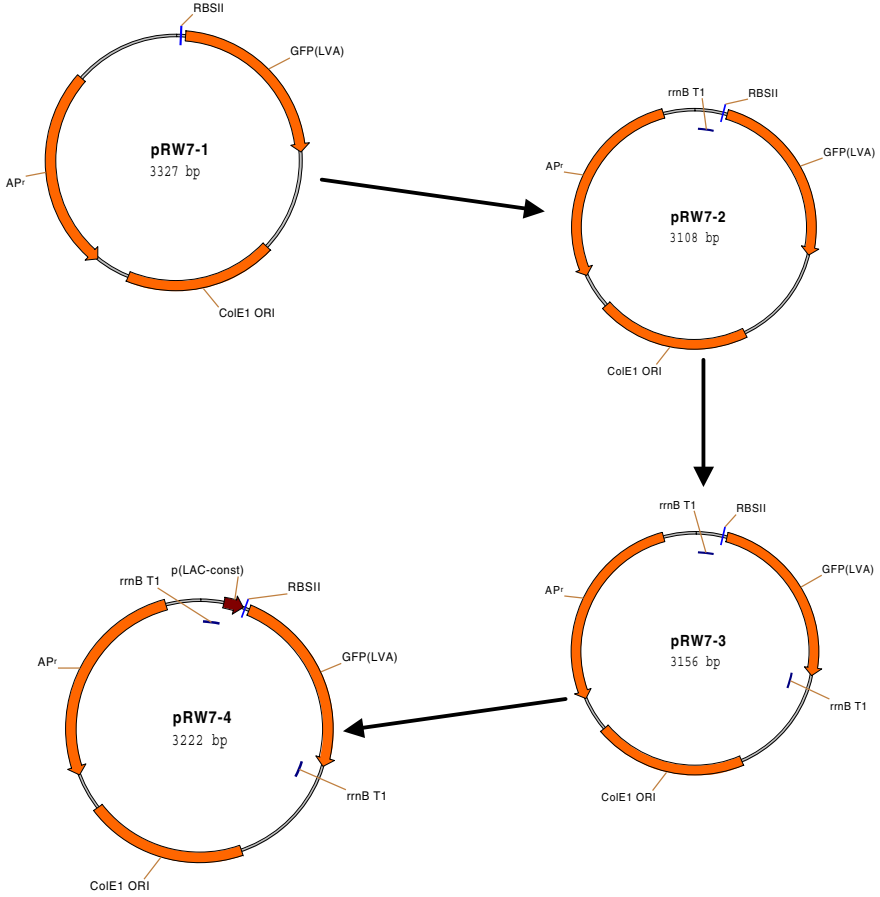


Fig. 2. Preliminary plasmids

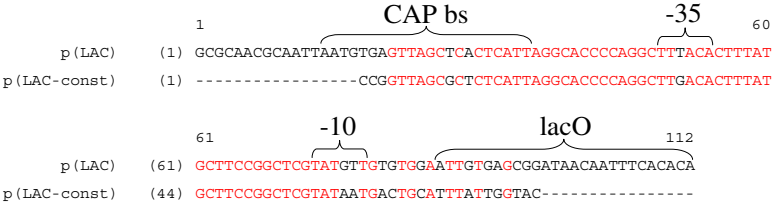


Fig. 3. Comparison of p(LAC) with p(LAC-const)

that were annealed using the same procedure as above. The plasmid pRW7-4 was transformed into *E. coli DH5 α* chemically competent cells. The construct consisting of p(LAC-cons) followed by GFP(LVA) was verified by detecting the fluorescence of the cells (data not shown).

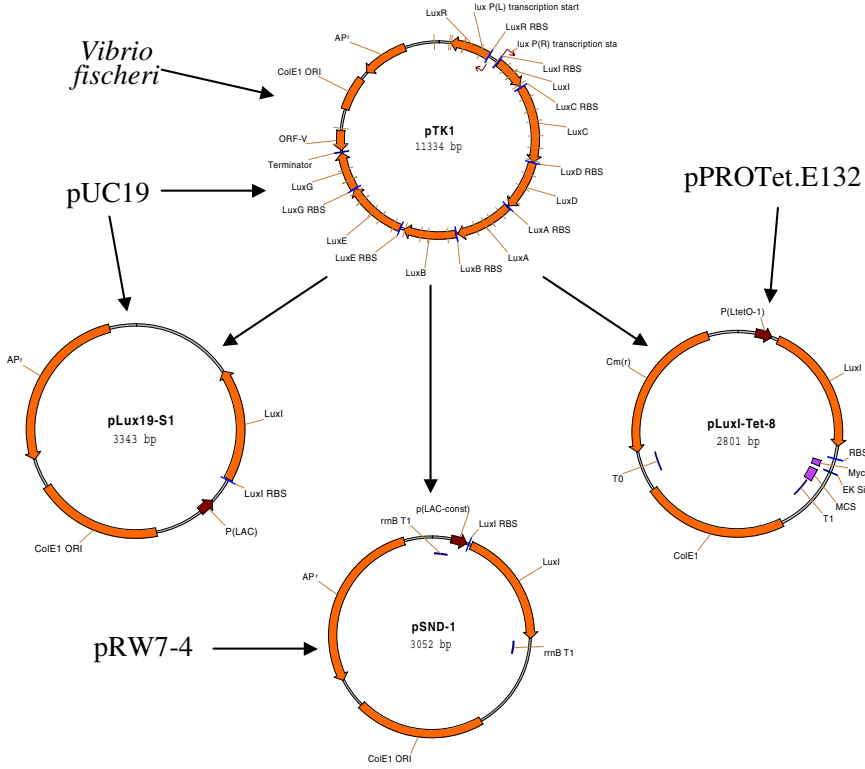


Fig. 4. Sender plasmids

4.2 Senders

We isolated individual components of the *Vibrio fischeri* system for further use. Plasmids described in this section are shown Figure 4. The *luxI* coding region was cloned and placed under control of the Lac promoter of the pUC19 plasmid. This was done by PCR of the pTK1 plasmid DNA using selected primers which included non-matching 5' *EcoRI* cut sites. Specifically, we performed a PCR reaction using forward primer 5'(agg↓aattcgaataaacgcaagggag)3' and reverse primer 5'(ccg↓aattccctataatatacttag)3', yielding the full length *luxI* coding sequence, including the ribosomal binding site, but with paired, distal *EcoRI* cut sites. PCR was performed using Life Technology High Fidelity PCR Supermix (25μl), 1μl of each primer, and 1μl of 300ng/μl pTK1 plasmid DNA. The reaction was denatured 5 minutes @94°C, followed by 30 cycles of denaturing 30 seconds @94°C, annealing 30 seconds @50°C, and extension 1 minute @70°C. Reaction products were verified by gel electrophoresis, and separated from primers using the Bio101 GeneClean spin protocol. The purified PCR product was digested with *EcoRI*, and ligated with prepared pUC19 vector, which had been cut with *EcoRI* and dephosphorylated with Amersham shrimp alkaline phosphatase.

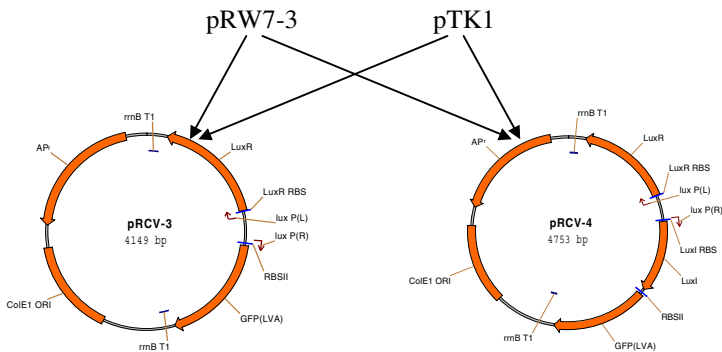


Fig. 5. Receiver plasmids

The resulting ligation was transformed into *E. coli DH5 α* and plated on LB AMP. The transformed colonies exhibited two distinct morphologies, clear, small colonies and opaque, large colonies. Six of each colony morphology were streaked, grown, and miniprep. Restriction digests and gel electrophoresis showed that the small colonies contained the *LuxI* gene in the correct, expressing, orientation. One such clone, pLuxI19-S1 was chosen for further study.

The same *EcoRI* digested *LuxI* PCR product was also similarly cloned into the Clontech pPROTET.E332 plasmid. This plasmid contains a Col-E1 ori, chloramphenicol resistance gene, and a TetO controlled promoter. The TetO promoter is inhibited by the TetR gene product, in the presence of the antibiotic tetracycline. The TetR gene is chromosomally carried in a special version of *E. coli*, which also carries the spectinomycin resistance gene. As a first step, the ligation reaction was transformed into subcloning efficiency DH5 α cells, grown up in LB chloramphenicol (50 μ g/ml). After verification of the correct insert, miniprep DNA was re-transformed into the TetR containing strain, which was then grown in LB spectinomycin chloramphenicol broth.

The PROTet system allows controlled expression of the inserted gene using varying amounts of a non-growth-inhibitory version of tetracycline, anhydrotetracycline (aTc). In this way, we can control expression of the *LuxI* gene, and hence the level of VAI, in these cells, through control over the aTc concentration.

The plasmid pSND-1 was constructed for constitutive expression of *luxI*, by removing the GFP(LVA) CDS from pRW7-4 and replacing it with *luxI* from pTK1. A PCR reaction using forward primer 5'(catgggtactctccgaataaagctttact-tactgtac)3' and reverse primer 5'(catgaagcttaacaacattaatttaagactgc)3' yielded the *luxI* coding sequence, including the ribosomal binding site. The PCR product was then ligated with a pRW7-4 *Acc65I*/*HindIII* digest, and transformed into chemically competent DH5 α .

4.3 Receivers

The receiver plasmid pRCV-3 was constructed using pRW7-3 as the plasmid backbone and by inserting the *luxR*_{P_LP_R region from pTK1 upstream of}

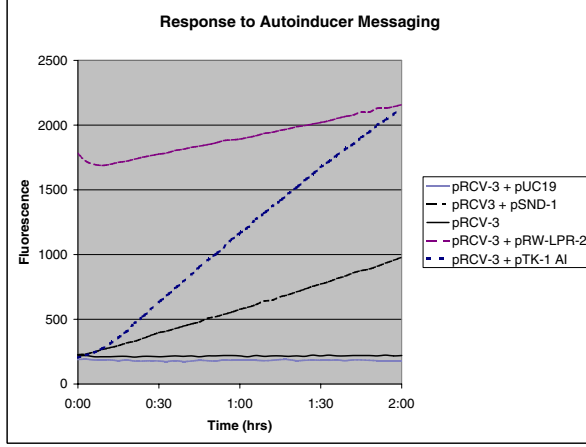


Fig. 6. Verification of communication constructs

GFP(LVA). We performed a PCR reaction using forward primer 5'(catgggtac-ctccggaataaaagctttacttacgtac)3' and reverse primer 5'(catgggtaccggccggtttatc-gactataacaaacc)3', yielding the $\text{luxR}P_LP_R$ region with *Acc65I* cut sites at both tails. The PCR product was then ligated into a pRW7-3 *Acc65I* digest, and the resulting colonies were screened by restriction mapping and partial plasmid sequencing to ensure that the insert was in the correct orientation.

The receiver plasmid pRCV-4 served as a control plasmid to verify the ability of the *lux* operon to exert positive control on the synthesis of GFP(LVA). The $\text{luxR}P_LP_R\text{luxI}$ region from pTK1 was extracted with a PCR reaction using forward primer 5'(catgggtacctccggaataaaagctttacttacgtac)3' and reverse primer 5'(ccttggtaccggccgaacaacattaatttaagactgc)3'. As above, the PCR product was then ligated into a pRW7-3 *Acc65I* digest, and the resulting colonies were screened by restriction mapping and partial plasmid sequencing to ensure that the insert was in the correct orientation.

5 Intercellular Signalling Experiments

5.1 Sending a Constant Cell to Cell Signal

Our first intercellular communications experiment involved the sending of a constant signal from one cell type to another. Cultures of *E. coli DH5 α* containing the pRCV-3 plasmid and the pSND-1 plasmids were grown separately overnight @37°C in LB AMP. A 96 well clear bottom plate was loaded with 200 μl of LB AMP in each well. 10 μl of pSND-1 cells were loaded horizontally to each cell, along with controls consisting of cells expressing GFP constitutively, *E. coli DH5 α* containing pUC19, and a series of wells containing extracted VAI (see below).

Vertically, 10 μl of cells containing the pRCV-3 construct were also loaded into each well. Thus, each well contained a variety of senders, and a uniform set of receivers. The plate was grown in a Biotek FL-600 fluorescent plate reader

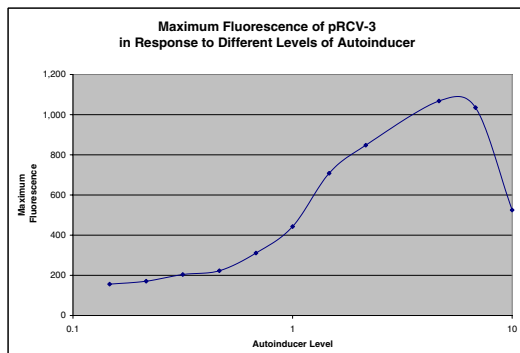


Fig. 7. The effect of different autoinducer levels on the maximum fluorescence attained.

for two hours, and read for fluorescence at the GFP(LVA) peak (excitation filter 485/20 nm, emission filter 516/20 nm). The results are shown in figure 6. Wells containing only the pRCV-3 cells, or with added pUC19 cells, showed no increase in fluorescence. The well containing pRCV-3 cells and pRW-LPR-2 cells (which express GFP(LVA)) served as a positive control. Wells containing the pRCV-3 cells plus extracted pTK1 autoinducer showed high, and increasing levels of fluorescence. Cells with pRCV-3 and pSND-1 showed the expected increase in fluorescence demonstrating successful cell to cell signalling.

5.2 Autoinducer Extraction and Characterization of the Receiver Module

The receiver plasmid pRCV-3 was further characterized by inducing the promoter with VAI extracted from cell culture. Cultures of *Vibrio fischeri* and of *E. coli* containing the pTK1 plasmid were grown overnight to stationary phase in GVM broth or LB AMP respectively @30°C which allows evaluation of their bioluminescence. After verification of light production, 100 ml of the cultures were centrifuged at 3300 g, and the supernatant collected. The supernatant was extracted with 10 ml of ethyl acetate by vigorous shaking in a separatory funnel for 10 minutes. The ethyl acetate extract (upper fraction) was separated and dried under vacuum. The resulting crude extract was redissolved in 1ml of DI water to provide 100x VAI extract.

We performed experiments to analyze the effectiveness of serial dilutions of the VAI extracts from pTK1 and *Vibrio fischeri* in inducing GFP expression of the pRCV-3 cells. Both the *Vibrio fischeri* and pTK1 extracts were about equally effective at inducing expression of the pRCV-3 promoter, as measured by GFP production. Figure 7 shows that increasing levels of autoinducer yielded increasing GFP expression by the receiver. High levels of the extract, however, were toxic to the cells, and resulted in relatively low fluorescence levels.

5.3 Sending Controlled Cell to Cell Signals

Finally, we placed the LuxI gene under control of the Tet promoter from the Clontech pPROTet system. The experiment is schematically represented in Fig-

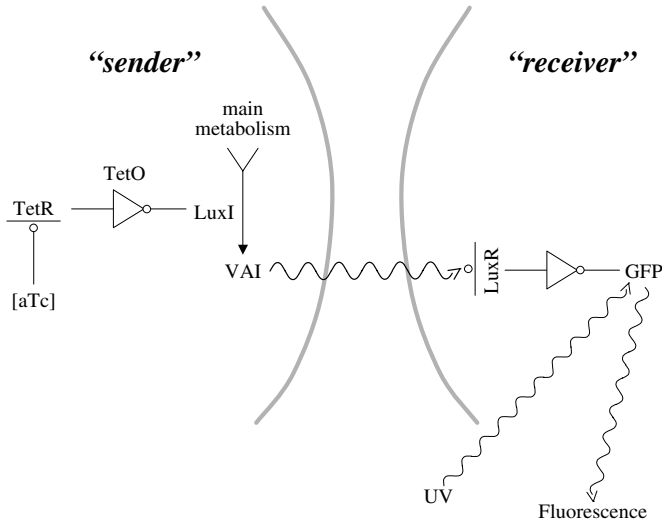


Fig. 8. Circuit diagram of gradient communications

ure 8. In one cell, the pLuxI-Tet-8 plasmid exerts controlled expression of the LuxI autoinducer synthesase using the Tet operon. The synthesase catalyzes the conversion of normal cellular metabolic products into VAI; thus, controlling the LuxI expression level controls the VAI production in the cells. The VAI produced within the cells migrates through the cell membrane of the sender, into the culture medium, and through the membrane of the receiver – a cell containing the pRCV-3 plasmid. There, it interacts with the N-terminal domain of the LuxR DNA binding protein product, disabling it from binding to the lux box binding site. The expression of the GFP reporter gene is enhanced, resulting in high levels of fluorescence.

The experiment involved the incubation of similar mixed cell cultures on 96 well clear bottom plates. One important difference was the culture medium – the pPROTET cells carry spectinomycin and chlormaphenicol resistance, while the pRCV-3 cells carry ampicillin resistance. The experiments were carried out by growing overnight cultures of both types of cells in the appropriate antibiotic containing medium, followed by centrifugation at 4000g to remove the medium, and resuspension to similar cell density in LB containing no antibiotics, so that both cell types could grow.

A similar arrangement of horizontally different pLuxTet senders (different colonies from the pPROTet/LuxI ligation reaction) and vertically similar receivers were loaded, including a set of wells containing no senders (“RCV only”) and a set of cells containing pTK1 VAI at levels previously shown to induce high level expression.

Figure 9 shows the results of this experiment after culturing the plate for four hours @37°C. As expected, the null wells showed no enhancement of fluorescence, while the 10x VAI positive control wells exhibited fluorescence. The three experiments labeled LuxTet4B7, LuxTet4B8, and LuxTet4B9 include senders

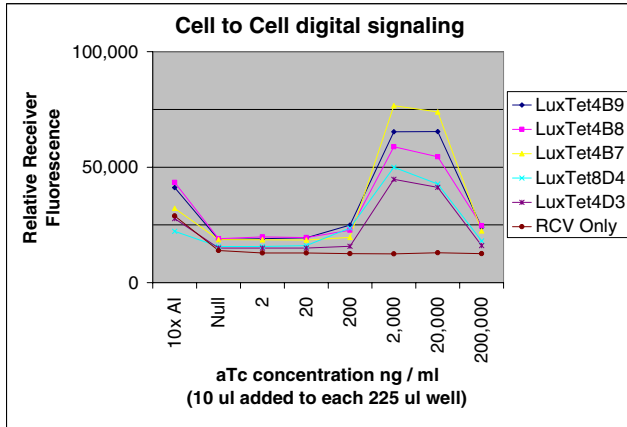


Fig. 9. The effect on the receiver of transmitting the message at different intensities

where the pLuxI-Tet-8 plasmid was transformed into BL21 – PRO cells, while the experiments labeled LuxTet4D3 and LuxTet8D4 include senders where the pLuxI-Tet-8 plasmid was transformed into *E. coli DH5 α* cells¹. In wells containing sender cells induced with aTc at levels below about 20ng/ml, only a small fluorescent response is exhibited by the receiver cells. In wells induced with aTc levels above 200ng/ml, a significant response was observed. Sufficiently high levels of aTc inhibited cell growth.

6 Conclusions

We have successfully isolated an important intercellular communication mechanism from a naturally occurring bacterial system, analyzed its components, and engineered its interfaces with standard genetic control and reporter mechanisms. While we have captured one such communication mechanism, realistic genetically controlled developmental systems will require perhaps dozens of such signals. The LasI/LasR system from *Pseudomonas aeruginosa* [5], for example, appears to encode a similar regulatory system, but one which uses a different, and non-cross reacting autoinducer, and a different structure homologous to the lux box. Isolation and characterization of such additional communication mechanisms will allow the construction of more complex multicellular systems.

A Extraction and Analysis of the Lux Operon Structure from Natural Constructs

A stab of *Vibrio fischeri* MJ1 was obtained from Fotodyne, Inc. and restreaked and grown @28°C on GVM plates (10g tryptone, 5g Difco casamino acids, 25g NaCl, 4g MgCl₂, 1g KCl, 15g agar per liter, pH 7.4) [31], and verified for light

¹ In BL21 – PRO cells, TetR (needed for controlled induction of the Tet promoter) exists on a plasmid, while in *DH5 α* TetR is part of the chromosomal DNA.

production. An overnight culture from a single colony was grown with vigorous shaking in liquid GVM medium @28°C to an OD of 2.5.

Genomic DNA was isolated from 100 ml of the overnight culture using a scaled up version of the CTAB procedure [2], hooked out of solution, ethanol reprecipitated, and dissolved in pH 8.0 TE 10:0.1.

A SalI restriction digest (cut site 5' g↓tctgac 3') of the genomic DNA was performed and run on a 0.8% TAE agarose gel to verify cutting and average fragment length. The digestion was performed in a 20μl reaction volume with 2μl (40 units) of SalI (NEB #138), 2μl of 10x NEBuffer SalI, supplemented with 0.2μl of 100x BSA, with 2μl of genomic DNA (300ng/μl). The digestion was carried out for 1 hour @37°C, followed by a denaturation @65°C for 20 minutes.

A similar SalI restriction digest of pUC19 plasmid DNA (NEB #304-1, (1μg/l)) was prepared and verified by 0.8% TAE agarose gel electrophoresis. pUC19 is a high copy number colE1 ori plasmid carrying the ampicillin resistance gene, and the LacZ β-galactosidase gene. Inserts into the pUC19 multiple cloning site disrupt the activity of the LacZ gene, and allow screening for insertions on XGAL plates using blue/white screening.

A ligation overnight @14°C of the SalI digests of pUC19 and the *Vibrio fischeri* genomic DNA was performed. The 20μl ligation reaction contained 2μl (800 Cohesive End Ligation Units) of T4 DNA Ligase (NEB #202), 2μl of 10x NEB T4 Ligase Buffer, 1μl of the SalI pUC19 digest DNA, and varying amounts (0.5, 1, 2, 4, 8μl) of the SalI genomic DNA digest.

The ligation mixture was transformed by 45 second heat shock @42°C into Life Technology subcloning efficiency *E. coli* DH5α [$F^- \Phi 80dlacZ-\Delta(lacZYA - argF)U269deoRrecA1endA1hsdR17(r_k^-.m_k^+)phoAsupE44\lambda - thi-1gyrA96relA1$] and spread on LB AMP (50mg/l) / XGAL (25mg/l) / IPTG (100μM) plates. The plates were incubated overnight @37°C to grow colonies, and evaluated for the optimal vector : insert ratio by inspection of blue/white colony ratios. The optimal ratio ligation mix (2μl of genomic DNA in the ligation) was spread onto 20 LB AMP plates, grown overnight @37°C, and further incubated for six hours at room temperature to allow expression of the heat sensitive Lux gene cassette. Plates were visually examined for luminescent colonies following dark adaptation. A single luminescent colony, labeled pTK1, was detected, and streaked out onto LB AMP plates.

A luminescent colony was used to inoculate 200 ml of LB AMP medium, and an overnight culture was grown with vigorous water bath shaking @37°C to OD 3.0. A standard Qiagen spin maxiprep was performed, yielding pTK1 plasmid DNA.

The pTK1 plasmid DNA was digested with EcoRI (as above with 2μl , 20 units, of NEB #101, (cut site 5' g↓aattc 3') and SalI restriction enzymes. The unrestricted plasmid DNA, as well as the restriction digests were run on 0.8% TAE agarose gels, along with samples of pUC19, pUC19 digested with SalI, and a 1 kb Biorad DNA ladder.

pUC19 digests with EcoRI and SalI showed the expected 2.86 kb fragment, while the undigested pUC19 showed several bands presumably corresponding to supercoiled variants of the circular plasmid.

pTK1 digests showed an EcoRI fragment of approximately 11.5 kb and two Sall fragments of sizes 2.7 kb and 9 kb. The 2.7 kb fragment was identified as the double cut pUC19 vector. The 9 kb fragment was identified as the luminescence causing vector insert.

The insert of pTK1 was sequenced with the Sanger dideoxy technique using ABI BigDye terminator ready reaction mix and primer walking. Initial sequences were primed with the M13 -47 forward primer 5'(cgccagggttttccagtcacgac)3' and the M13 -48 reverse primer 5'(agcggataacaatttcacacagga)3'. Subsequent primer sequences were determined by choosing 18-22 mers from about 500 bp into the previous sequence. Primer sequences were chosen with approximately 50% gc content, typically ending (3') with two or more gc bases to act as clamps. Reverse complement sequences were also chosen from about 250 bases into the new sequence using similar criteria, to prime a reverse direction verification sequence. The sequencing reactions were carried out in a 25 μ l volume, containing 4 μ l of ABI BigDye Ready Reaction Mix, 4 μ l of Sequencing buffer (200mM Tris-HCl, 5mM MgCl₂, pH 9), 1 μ l of primer, and 1 μ l of pTK1 plasmid template. The sequencing reactions were run on an MJ PTC-200 thermal sequencer, with a program consisting of a denaturing step of @95°C for 10 minutes, followed by 30 cycles consisting of 10 seconds @94°C, 5 seconds @50°C, and 4 minutes @50°C. Sequencing reactions were then held until use @4°C in the cycler. The sequencing reaction mix was gel filtered in a Princeton Separations sepharose column, dried in a Speedvac, and resuspended in 25 μ l of ABI template suppression buffer. After vortexing and spin down, the resuspended product was denatured @95°C for 1 minute, and snap cooled on wet ice. The product was transferred to septum covered tubes and inserted into the ABI 310 sequencer. Samples were run with a 61 cm capillary, filled with ABI POP-6 sequencing gel, held @50°C, and with a 3KV, 60 second capillary injection, followed by a 12.2KV 120 minute electrophoresis run.

Sequences were proofread with ABI Sequence Manager software, and several omitted bases of the initial sequences were manually corrected. Corrected sequences were assembled using the ABI Autoassembler software, yielding a complete sequence which was again proofread. Two additional sequences were run to verify questionable sequence calls. The final insert was determined to be 8654 bases long, and has been submitted to GENBANK as accession AF170104 [21].

The resulting sequence was compared against other known sequences in the GENBANK NR database, and found to be essentially identical to previously reported sequences for the *Vibrio fischeri* MJ-1 strain, for those portions which had been previously sequenced. Specifically, the reported sequence is completely identical in those regions reported in GENBANK M25751 [7] for strain MJ-1, and differing only slightly from the earlier reports from sequencing this strain in GENBANK Y00509 [13]. Specifically, these differences are a missing triplet gtt at base 891, a short, recovered, frame shift mutation at bases 904 - 910, and a substituted cgc for a gcg triplet at base 1137. Given the identical sequence reported in M25751, these differences probably represent mutations in their copy of the original MJ-1 sequence.

Related *Vibrio fischeri* strains show high homology in these areas as well. Entry M19039 [6], sequencing from ATCC 7744 (type strain) shows 23 point

mutations relative to our sequencing of this area. M96844, [17], sequenced from the squid symbiot ES114 strain, shows a relatively distant but still quite close homology.

References

1. H Abelson, D Allen, D Coore, C Hanson, G Homsy, TF Knight, R Nagpal, E Rauch, GJ sussman, and R Weiss, *Amorphous computing*, Tech. Report AI Memo No. 1665, MIT Artificial Intelligence Laboratory, 1999.
2. FM Ausubel, R Brent, RE Kingston, DD Moore, JG Seidman, JA Smith, and K Struhl, *Short protocols in molecular biology*, Wiley, 1999.
3. D Coore, *Botanical computing: A developmental approach to generating interconnect topologies on an amorphous computer*, Ph.D. thesis, Massachusetts Institute of Technology, 1998.
4. D Coore, R Nagpal, and R Weiss, *Paradigms for structure in an amorphous computing*, Tech. Report AI Memo No. 1614, MIT Artificial Intelligence Laboratory, 1997.
5. T deKievit, PC Seed, J Nezezon, L Passador, and BH Iglewski, *Rsal, a novel repressor of virulence gene expression in pseudomonas aeruginosa*, J. Bacteriol **181** (1999), 2175–2184.
6. JH Devine, C Countryman, and TO Baldwin, *Nucleotide sequence of the luxR and luxI genes and structure of the primary regulatory region of the lux regulon of vibrio fischeri atcc7744*, Biochemistry, vol. 27, 1988, GENBANK M19039 (strain ATCC 7744), pp. 837–842.
7. JH Devine, GS Shadel, and TO Baldwin, *Identification of the operator of the lux regulon from the vibrio fischeri strain atcc7744*, Proc. Natl. Acad. Sci. USA, vol. 86, 1989, GENBANK M25751 (strain MJ-1) and M25752 (strain ATCC 7744, subset of M19039), pp. 5688–92.
8. M Dworkin, *Cell-cell interactions in the myxobacteria*, Symp. Soc. Gen. Microbiol., vol. 25, 1973, pp. 135–147.
9. M Dworkin and D Kaiser, *Cell interactions in myxobacterial growth and development*, Science **230** (1985), 18–24.
10. A Eberhard, AL Burlingame, C Eberhard, GL Kenyon, KH Nealson, and NJ Openheimer, *Structural identification of autoinducer of photobacterium fischeri luciferase*, Biochemistry, vol. 20, 1981, pp. 2444–2449.
11. M Elowitz and S Leibler, *A synthetic oscillatory network of transcriptional regulators*, Nature **403** (2000), 335–338.
12. J Engebrecht, KH Nealson, and M Silverman, *Bacterial bioluminescence: isolation and genetic analysis of the functions from vibrio fischeri*, Cell, vol. 32, 1983, pp. 773–781.
13. J Engebrecht and M Silverman, *Nucleotide sequence of the regulatory locus controlling expression of the bacterial genes for bioluminescence*, Nuc. Acids Res., vol. 15, 1987, GENBANK Y00509 (strain MJ-1), pp. 10455–10467.
14. C Fuqua and A Eberhard, *Signal generation in autoinduction systems: synthesis of acylated homoserine lactones by LuxI-type proteins*, 211–230, GM Dunny and Winans, Washington, DC, 1999, pp. 211–230.
15. WC Fuqua, S Winans, and EP Greenberg, *Quorum sensing in bacteria: The luxR-luxI family of cell density-responsive transcriptional regulators*, J. Bacteriol **176** (1994), 269–275.

16. T Gardner, R Cantor, and J Collins, *Construction of a genetic toggle switch in escherichia coli*, Nature **403** (2000), 339–342.
17. KM Gray and EP Greenberg, *Sequencing and analysis of luxr and luxI, the luminescence regulatory genes from the squid light organ symbiont vibrio fischeri es114*, Molecular Marine Biology and Biotechnology, vol. 1, 1992, GENBANK M96844 (strain ES114), pp. 414–419.
18. EP Greenberg, *Quorum sensing in gram-negative bacteria*, ASM News, vol. 63, 1997, pp. 371–377.
19. Thomas F. Knight Jr. and Gerald Jay Sussman, *Cellular gate technology*, First International Conference on Unconventional Models of Computation (CS Calude, J Casti, and MJ Dinneen, eds.), Springer-Verlag, 1998, pp. 257–272.
20. D Kaiser, *Regulation of multicellular development in myxobacteria*, Microbial Development (1984), 197–218.
21. TF Knight and N Papadakis, *Vibrio fischeri lux operon sali digest*, GENBANK AF170104 (strain MJ-1), 1999.
22. S Lissner and H Margalit, *Compilation of escherichia coli mrna promoter sequences*, Nuc. Acids Res. **21** (1993), no. 7, 1507–1516.
23. Harley H. McAdams and Adam Arkin, *Simulation of prokaryotic genetic circuits*, Annu. Rev. Biophys. Biomol. Struc. **27** (1998), 199–224.
24. R Nagpal, *Organizing a global coordinate system from local information on an amorphous computer*, Tech. Report AI Memo No. 1666, MIT Artificial Intelligence Laboratory, 1999.
25. A Orosz, I Boros, and P Venetianer, *Analysis of the complex transcription termination region of the escherichia coli rrnB gene*, Eur. J. Biochem, vol. 201, 1991, pp. 653–659.
26. E Rauch, *Discrete, amorphous physical models*, Master’s thesis, Massachusetts Institute of Technology, 1999.
27. EG Ruby and KH Nealson, *Symbiotic association of photobacterium fischeri with the marine luminous fish monocentris japonica: a model of symbiosis based on bacterial studies*, Biol. Bull **151** (1976), 574–586.
28. AM Stevens and EP Greenberg, *Transcriptional activation by luxr, in cell-cell signaling in bacteria*, Cell-Cell Signaling in Bacteria (GM Dunny and Winans, eds.), American Society for Microbiology, 1999.
29. Ron Weiss and George Homsy, *Toward in-vivo digital circuits*, Dimacs Workshop on Evolution as Computation (Princeton, NJ), January 1999.
30. Ron Weiss, George Homsy, and Radhika Nagpal, *Programming biological cells*, Eighth International Conference on Architectural Support for Programming Languages and Operating Systems, Wild and Crazy Ideas Session (San Jose, California), October 1998.
31. MR Winfrey, MA Rott, and AT Wortman, *Unraveling dna: Molecular biology for the laboratory*, Prentice Hall, 1997.

Successive State Transitions with I/O Interface by Molecules

Ken Komiya¹, Kensaku Sakamoto, Hidetaka Gouzu, Shigeyuki Yokoyama, Masanori Arita³, Akio Nishikawa², and Masami Hagiya

¹ Department of Biophysics and Biochemistry,

² Department of Information Science,

Graduate School of Science, University of Tokyo,

7-3-1 Hongo Bunkyo-ku, 113-0033 Tokyo, JAPAN.

{komiya,sakamoto,yokoyama}@biochem.s.u-tokyo.ac.jp

{nisikawa,hagiya}@is.s.u-tokyo.ac.jp

³ Electrotechnical Laboratory

1-1-4 Umezono, Tsukuba-shi, 305-8568 Ibaraki, JAPAN.

arita@etl.go.jp

Abstract. This paper reports three experimental achievements in our computation model based on ‘whiplash’ reactions. We first show that a single-stranded DNA (ssDNA) can serve as an independent machine by using a solid support technique. Second, we show how to append an arbitrary sequence, e.g. a transition state or a PCR primer, to the 3'-end of a molecular machine, thus realizing its I/O interface. Finally we demonstrate the successive state transitions for several steps on solid phase with I/O.

1 Introduction

Autonomous DNA computing assumes that each molecule works not only as a data carrier but also as a microscopic computing unit. Since we introduced Whiplash model, a computation model based on whiplash reactions (Figure 1 [4], several advances on our model have been reported in different directions. In the theoretical side, Winfree showed the implementation of GOTO programs and the efficient solution of the directed Hamiltonian path problem [10]. In the experimental side, we showed the more efficient transition steps using an isothermal reaction [9].

The isothermal conditions have made the successive state transitions more realistic than do the common thermal schedules for PCR [9]. The normal PCR cycle comprises the three steps of denaturation, annealing, and polymerization, each performed at a different temperature. In contrast, the isothermal program intends to perform these three steps simultaneously, thus coupling the dissociation of the ‘current state’ sequence from the transition table with the polymerization of the ‘next state’ sequence (Figure 1). This coupled reaction was found to actually occur at 80 °C, with an optional use of 64 °C that facilitates a hairpin formation.

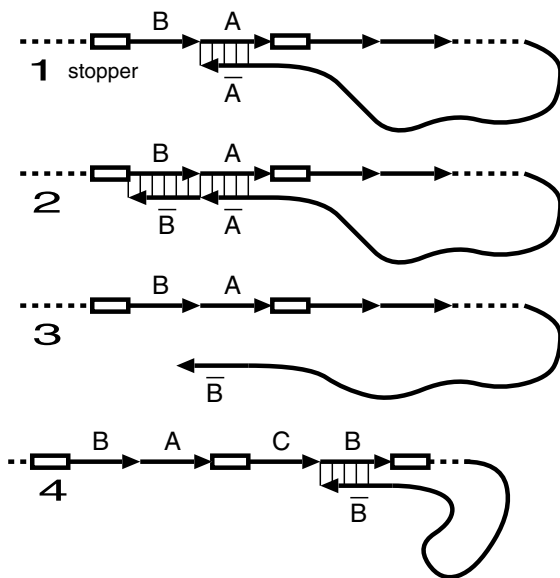


Fig. 1. Whiplash model is the first to implement a program, or state transitions, into ssDNA. (1) The current state **A** anneals to a transition table forming a hairpin structure. (2) At its 3'-end, the next state **B** is appended by polymerization, which stops at the stopper sequence (white box). (3) Denaturation. (4) The new state **B** anneals to another position in the transition table.

In this paper, we show the **successful multiple transition steps on solid phase** in Whiplash model. We also report the successful implementation of **input and output (I/O) interface** in the model.

The implementation of I/O interface is a significant progress from the already reported experimental results. From the beginning of DNA computing, molecules have been used as a memory device in realistic computation models. Although 'DNA as memory' is the original paradigm shown by Adleman and Lipton, it confined DNA computing to the 'single-instruction, multiple-data (SIMD)' model. In order to realize 'multiple-instruction, multiple-data (MIMD)' model, it is necessary to implement both a program and I/O interface in DNA sequences. Whiplash model is the first realistic model to support both with DNA.

Independence of each molecular machine is a prerequisite for an autonomous molecular computation. Although each DNA molecule can, in theory, work as one machine in our model, DNA molecules were not immobilized and could freely interact in the experiment of our previous paper [4]. In this paper, we employ the surface-based approach to prevent an intermolecular reaction. Thus, Whiplash model made a major progress toward realistic MIMD computation.

Reactions on solid phase naturally necessitate I/O interface for a molecular machine. Since DNA molecules are immobilized on solid phase, we cannot

directly analyze the whiplash reaction by gel electrophoresis, as we did previously [9]. In addition, the subsequence at the 3'-end of the molecule cannot be used as a PCR primer-binding site, because the 3'-end sequence always has a complementary counterpart in the transition table, and the base pairing between them inhibits PCR amplification. The way to avoid this difficulty is to append a 'readout' sequence to the 3'-end in the course of the successive transitions (Figure 2). The elongated molecules can be subjected to PCR amplification with the readout sequences as primers, thus reporting the current state of the machine. Note here that, under some experimental conditions, only a minor population will undergo this sequence extension, and the remaining molecules can continue further transition reactions.

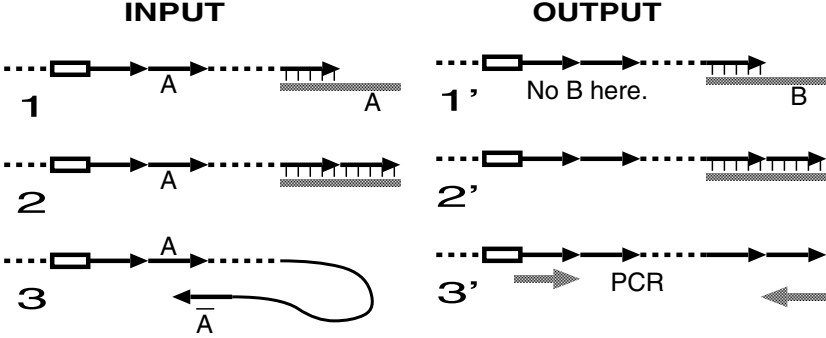


Fig. 2. The I/O interface. (1) The input oligomer (shaded bar) anneals to the 3'-end of the molecular machine. (2) The 3'-end is extended, forming the initial state \bar{A} . (3) Start transition steps. Note that we cannot set \bar{A} when the whole ssDNA is synthesized, because transitions would start in the preparation (PCR) step. (1')(2') For the output reaction, the readout sequence **B** will be copied to the 3'-end of the machine. (3') Transition stops, because **B** does not appear in the transition table. The machine can be amplified by PCR.

Consequently, we implemented I/O interface with molecules, which can:

- probe the status of transitions, and
- start the transition from an internal state.

We might also realize the data transfer between molecular machines with this technique.

This paper is organized as follows. Our experiments and their results are introduced in Section 2. We discuss the possibility of increasing transition steps, and its accompanying experimental hurdles in Section 3.

2 Experiments

2.1 Materials and Methods

DNA. The DNA sequences used here (Table 1) were designed using a genetic algorithm package, GENESIS. The details of this sequence design will be presented elsewhere [1]. Oligonucleotides, including 5'-biotinylated ones, were commercially synthesized by Amersham Pharmacia Biotech (Tokyo, Japan). Hereafter, DNA sequences and oligomers are represented with bold letters.

Table 1. DNA sequences output by the genetic algorithm

no.	sequence
0	CCGTCTTCTTCTGCT
1	TTCCCTCCCTCTCTT
2	CGTCCTCCTCTTGTT
3	CCCCTTCTTGTCTT
4	TGCCCCTCTTGTTCT
5	CTCCTCTTCCTTGCT
6	CTTCTCCCTTCCTCT
7	CCTTCCTTCCTCTT
8	TCCCCTTGTGTGTGT
9	GAGAGAGAGCCCCCTATCC
10	GAAGAGAAGGGCACCCCTCC
11	GGGAAGGGACGCAACACCAC

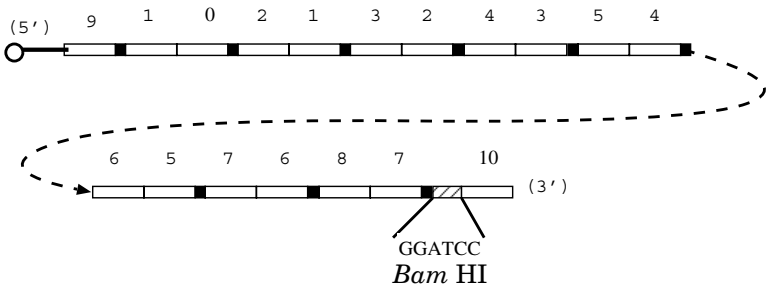


Fig. 3. Tran7 is represented in a concatenated form of states.

Solid Support Technology. The molecular machine **Tran7**, in the form shown in Figure 3, was amplified by PCR with the 5'-biotinylated primer **9** and the

primer $\overline{10}$, whose sequence is complementary with that of **10**, and was then immobilized to streptavidin-coated beads (Dynal) according to the supplier's protocol. Conversion of **Tran7** into the single-stranded form was done by alkaline treatment.

I/O Interface. An arbitrary sequence can be appended to the 3'-end of the state machine by polymerization using an appropriate oligomer as the template. This polymerization was performed under the conditions similar to the whiplash reaction [9]; the reaction buffer contains only dATP, dGTP, and dCTP but not dTTP, and the thermal schedule for 'input' reaction was 94 °C for 30 sec, 80 °C for 30 sec, and 64 °C for 20 min, with the addition of *rTaq* DNA polymerase (Toyobo, Tokyo, Japan) just before the incubation at 80 °C.

For starting the successive transitions, an initial state was set on **Tran7** by appending the sequence complementary to a state sequence (from **1** to **8**) using an appropriate input oligomer. After this 'input' reaction, the input oligomer was removed by alkaline treatment from **Tran7**, and then **Tran7** was subjected to successive whiplash reactions.

For readout of the results of computation, the output oligomers ($\overline{11-1-0}$, $\overline{11-2-1}$, \dots to $\overline{11-8-7}$) are to be added after the completion of successive transitions. On the other hand, for probing the status of the machine (**Tran7**), the output oligomers were added, into the reaction mixture, in the course of successive transitions. In both cases, the 'readout' sequence (**11**) is appended to the 3'-end of (a subset of) the molecules.

Successive Transitions (whiplash reactions). **Tran7** with the initial state, immobilized on the beads, was added to the buffer provided by the supplier (Toyobo) containing *rTaq* DNA polymerase (5 units), dATP, dGTP, and dCTP (0.2 mM each). State transition was performed in a 25 μ l reaction, which contains 5 pmol molecules (**Tran7**). The thermal schedule is as follows:

- initial incubations at 80 °C for 1 min
- add *rTaq* DNA polymerase
- 94 °C for 30 sec
- gradual cooling (in 2 min) to 64 °C
- 15 reaction cycles
 - 64 °C for 30 sec
 - shift up to 80 °C in 1 min
 - 80 °C 5 min

2.2 Results

We developed an isothermal technique to perform successive whiplash reactions [9]. Since we already achieved the efficient transitions up to 2 steps, more steps were tried in the present study. The employed DNA molecule (**Tran7**) is long enough to easily interact intermolecularly. For the purpose of preventing

this intermolecular interaction, each **Tran7** was immobilized on a solid surface. The initial state **1** was appended with the input oligomer (**1–0–10**) to the 3'-end of the immobilized molecule, which was then allowed to perform the whiplash reactions.

The whiplash reaction consists of 15 cycles, each of which includes the incubation phase at 64 °C, and the transition phase at 80 °C. Note that the incubation phase is only to facilitate hairpin-structure formation. This operation is inherently different from a normal PCR reaction, in which polymerization and denaturation occur at a different temperature. Therefore, our reaction can be called an isothermal-reaction.

During the successive transitions, each output oligomer was added to the reaction mixture for the status probing (Figure 4). The oligomer hybridized with the 3'-end of the machine after the completion of the specific transition step. Note here that this output reaction competes with the whiplash reaction for the next transition; the extent of this competition is not yet examined.

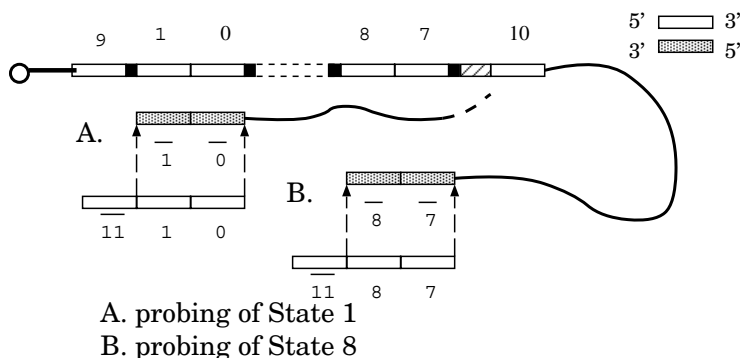


Fig. 4. The output oligomer $\overline{11}$ –1–0 appends the readout sequence **11** to the 3'-end, and it probes the state **1**. In the same way, the output oligomer $\overline{11}$ –8–7 works for probing the state **8**.

For converting the state machine into the double-stranded form, dTTP was added to the reaction mixture after the status probing. This conversion is necessary to use a restriction enzyme (*Bam*HI) for cutting off the part polymerized during successive transitions. The cut-off part was then amplified by PCR, with primers **10** and $\overline{11}$, in order to detect it on a 8% polyacrylamide gel stained with ethidium bromide (Figure 5).

The bands with the expected mobilities appeared up to 4 successive steps (lanes 0 – 4), and were significantly detected for the 5th and 6th steps, with major bands due to the unexpected products. The band for the 7th step was not observed.

The unexpected products in the 5th and 6th steps were subjected to sequence analysis, and were found to comprise $\overline{11}$ – $\overline{6}$ – $\overline{5}$ – $\overline{10}$ for the 5th step, and $\overline{11}$ – $\overline{2}$ – $\overline{1}$ – $\overline{0}$ – $\overline{10}$ for the 6th.

The reason for the occurrence of these sequences is probably hybridization between the extended part of the machine and PCR primers (or the remains of output oligomers), due to accidental similarities in their sequences. This was a pitfall in our sequence design.

In order to show that our technique enables the state machine to start from any state, we appended sequence $\overline{3}$ – $\overline{4}$, forcing the machine to start from the 4th state. We thus succeeded in skipping the first 4 steps, and in performing the following 4 transitions up to state **8** (data not shown).

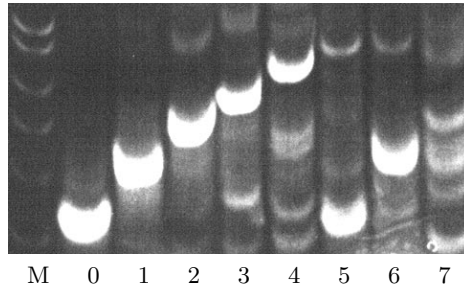


Fig. 5. Gel electrophoresis of the transition products. The transition was successful up to the 4th step. Unexpected short major bands appear in the 5th and 6th steps.

3 Discussion

3.1 Benefits from I/O Interface

We realized the appending of any transition state or any readout sequence at the 3'-end of the state machine. This implementation of I/O interface greatly improved the practicality of Whiplash model. A typical design of readout process for previous DNA computers has been to hybridize a marker DNA to a specific position which is pre-designed in DNA sequences. That is, what can be read is *fixed* by the initial sequence design. In our approach, on the other hand, reading or outputting an arbitrary state became possible. We list some of its applications.

Starting from an Internal State. By appending an input sequence, we can start the state transition from any internal state, and can skip some transitions.

Probing Transition States. By appending a PCR primer-binding site, we can check whether a certain state is transited. Note that appending of a primer-binding site does not alter the execution of other machines. This means that we can *observe* the transitions, by not stopping the execution of the unprobed machines.

Data Transfer between Molecules. With the I/O technique, we could feed the output of one machine as the input of another. We could also preserve the computed data for the execution of different machines. This is a kind of ‘DNA to DNA computations’ [11], where information on a DNA molecule is transferred to another molecule, according to a logic arbitrarily given [11][12][13]

3.2 Reaction on Solid Phase

Reaction using solid phase techniques has been used in DNA computing, mainly for solving NP-complete problems [6][5]. The scalability and the potential for automation are usually referred as the advantages of the solid support, because DNA is used only as a memory device.

On the other hand, the importance of solid support for whiplash reactions is guaranteeing the independence of each molecule. In this sense, our idea of this use of the surfaces is hinted by a report on an *in vitro* selection of RNA enzymes (ribozymes) [2]. A ribozyme is a kind of the sophisticated molecular machines that can be artificially produced [3].

3.3 Sequence Design

Since the melting temperature of DNA molecules depends on their GC content, it is best to uniformly distribute GC in sequences. For this reason, we set the GC content of sequences to be the same. More important is the avoidance of unexpected polymerization from the 3'-end. For this reason, this part should not completely hybridize with other sequences.

However, the sequence design satisfying these conditions became extremely difficult, especially when:

- we can use only three bases, and
- we fix the number of GCs in each sequence.

One solution to get over this hardness is to enhance the genetic alphabet by using artificial bases such as isoguanosines (iG) or 5-methylisocytidines (iC). We do not intend, however, to fully mix these bases with natural bases. We only need to introduce a single artificial base near each 3'-end, because the polymerization of misproducts does *not* start unless the 3'-end completely hybridizes with a ‘false’ site. Therefore, a single iC (or iG) is enough to prevent an unexpected extension. Although artificial bases are not used in our experiment of this paper, the prospect of their special use is shown in our previous work [9].

3.4 Increasing the Number of Transition Steps

The reliability and the efficiency of intramolecular reaction may have been doubted, but we demonstrated the successful multi-step transitions. In this paper, we introduced three experimental landmarks in Whiplash model:

- state transition on solid phase,
- implementation of an I/O interface,
- state transition for several steps.

In whiplash reactions, the transition from the current state to the next one competes with the ‘back annealing’ of the current state to the previous position in the transition table. This difficulty was overcome with the isothermal conditions, developed previously [9].

Several successive transitions were expected to confer other difficulties. One is that the sequences added at the 3'-end during the transitions have complementary counterparts in the transition table. Therefore, the longer becomes the state machine, the more chance it has to take a hairpin structure, which prohibits further transitions in the present design of the transition table.

However, the drastic reduction in efficiency after the 4th step, as shown above, was probably caused by the mis-annealing of the 3'-end subsequence to inappropriate PCR primers or oligomers for output. It should be made clear which type of error is responsible for this result: an error during the state transition, or an error during the PCR amplification for readout. Although either possibility must be separately considered in the design of DNA sequences, these possibilities may at least be attributed to the composition biased toward C and T. (Note that we do not use A in sequences.) They were designed to contain very few G, in order not to hybridize with each other, but this bias seems to have induced the unexpected similarity between 3'-termini of sequences. Careful re-design of sequences with more G is now ongoing. We consider that the transition for 10 steps will be possible with ‘good’ sequences.

Another difficulty relates to the multiple occurrence of a state in the table, making the machine go along branched paths. This makes the ‘back annealing’ described above even more serious. This undesired effect of the ‘back annealing’ was analyzed in terms of statistical thermodynamics by John A. Rose and Russell J. Deaton (University of Memphis).

Acknowledgment. This work is supported by the Japan Society for the Promotion of Science “Research for the Future” Program (JSPS-RFTF 96I00101).

The fifth and seventh authors are supported by Grant-in-Aid for Scientific Research on Priority Area “Genome Science” from Ministry of Education, Science, Sports and Culture, Japan.

References

1. Arita, M., Nishikawa, A., Hagiya, M., Komiya, K., Gouzu, H., and Sakamoto, K.: Improving Sequence Design for DNA Computing, pp.875–882, *Proceedings of GECCO2000*, 2000.
2. Bartel, DP. and Szostak, JW.: Isolation of New Ribozymes from a Large Pool of Random Sequences, *Nature* **261**, pp.1411–1418, 1993.
3. Ellington, AD., Robertson, MP., James, KD., and Cox, JJ.: Strategies for DNA Computing, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* **48**, pp.173–184, 1999.
4. Hagiya, M., Arita, M., Kiga, D., Sakamoto, K., and Yokoyama, S.: Towards Parallel Evaluation and Learning of Boolean mu-formulas with Molecules, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* **48**, pp.57–72, 1999.
5. Liu, Q., Wang, L., Frutos, AG., Condon, AE., Corn RM., and Smith LM.: DNA Computing on Surfaces, *Nature* **403**, pp.175–179, 2000.
6. Morimoto, N., Arita, M., and Suyama, A.: Solid Phase DNA Solution to the Hamiltonian Path Problem, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* **48**, pp.193–206, 1999.
7. Oghihara, M. and Ray, A.: Simulating boolean circuits on DNA computers, *Proc. 1st International Conference of Computational Molecular Biology* (ACM Press), pp.326–331, 1997.
8. Piccirilli, JA., Krauch, T., Moroney, SE., and Benner, SA.: Enzymatic incorporation of a new base pair into DNA and RNA extends the genetic alphabet, *Nature* **343**, pp.33–37, 1990.
9. Sakamoto, K., Kiga, D., Komiya, K., Gouzu, H., Yokoyama, S., Ikeda, S., Sugiyama, H., and Hagiya, M.: State Transitions by Molecules, *Biosystems* **52**, pp.81–91, 1999.
10. Winfree, E.: Whiplash PCR for $O(1)$ computing, *Proc. 4th DIMACS Workshop on DNA Based Computers*, pp.175–188, 1998.
11. Landweber, LF., Lipton, RJ., and Rabin, MO.: DNA²DNA Computations: A potential ‘Killer App’?, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* **48**, pp.161–172, 1999.
12. Suyama, A., Nishida, N., Kurata, K., and Omagari, K. :a poster abstract for *RECOMB2000*, 2000.
13. Brenner, S., Williams, SR., Vermaas, EH., Storck, T., Moon, K., McCollum, C., Mao, JI., Luo, S., Kirchner, JJ., Eletr, S., DuBridge, RB., Burcham, T., and Albrecht, G.: In vitro cloning of complex mixtures of DNA on microbeads: physical separation of differentially expressed cDNAs, *Proceedings of National Academy of Science USA* **97**, pp.1665–70, 2000.

Solution of a Satisfiability Problem on a Gel-Based DNA Computer

Ravinderjit S. Braich, Cliff Johnson, Paul W.K. Rothmund, Darryl Hwang,
Nickolas Chelyapov, and Leonard M. Adleman

University of Southern California
Laboratory for Molecular Science
835 West 37th Street, SHS 172
Los Angeles, CA 90089

Abstract. We have succeeded in solving an instance of a 6-variable 11-clause 3-SAT problem on a gel-based DNA computer. Separations were performed using probes covalently bound to polyacrylamide gel. During the entire computation, DNA was retained within a single gel and moved via electrophoresis. The methods used appear to be readily automatable and should be suitable for problems of a significantly larger size.

1 Introduction

In 1994 Adleman demonstrated the use of DNA molecules as a means of solving computational problems [1]. The vast parallelism which computing with molecules potentially affords has led to speculation that molecular computers might be suitable for attacking problems that have resisted conventional methods [1,2,3,4,5,6,7,8,9].

While the theory of molecular computation has developed rapidly, the practice of molecular computation has not kept pace. Although several groups have investigated molecular computation in the laboratory [1,10,11,12,13,14,15,16,17,18,19,20], no problem has yet been solved that most humans would find daunting. This paper reports on the progress of our group in attempting to create a molecular computer capable of solving problems that would be beyond the range of humans without the aid of electronic computers.

The model of computation described here is related to the previously described Sticker Model [21]. The Sticker Model uses two basic operations for computation: separation based on subsequence and application of stickers. In the experiment reported here, only separations were used.

Our initial approach to separation involved the incubation of a solution containing a DNA library with probes attached to a solid support (beads or filters). Molecules with appropriate subsequences hybridized to probes and were captured; molecules without such subsequences were removed by washing. Captured molecules were released back into solution by heating in new buffer. This seemingly straightforward approach did not work well in our hands. First, the hybridization of DNA in a 3-dimensional solution with probes immobilized on

a 2-dimensional solid support was unacceptably slow. Second, molecules that should have been retained were lost at an unacceptably high rate during washing. Third, efficient release of captured molecules was achieved only with the use of a large volume of buffer, with the result that the DNA library became increasingly dilute as the computation progressed.

Recently, Mosaic Technologies (Boston, MA) introduced the AcryditeTM phosphoramidite for modifying DNA molecules at the 5'-end during chemical synthesis. Like an acrylamide monomer, the AcryditeTM phosphoramidite carries a reactive ethylene functionality. Hence, standard methods can be used to copolymerize AcryditeTM-modified DNA probes into polyacrylamide gels—covalently linking the probes to the gel matrix. Because a gel is a nearly liquid environment, AcryditeTM-linked DNA probes apparently approximate probes in a solution. This gives one benefits of a solid-support-based system while still retaining characteristics of a solution-based system. In particular, the capture rate of molecules with the proper subsequence is improved, presumably because the 3-D to 2-D transition is mitigated. In addition, DNA can be moved by electrophoresis rather than transported by the movement of buffer; hence the problems of volume increase and library dilution are solved.

Our pilot separation experiments with AcryditeTM were adequate to warrant testing the technology in a DNA-based computation on a 3-SAT problem. We chose to solve the 6-variable 11-clause formula

$$\Phi = \begin{aligned} & (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_2 \vee \neg x_3 \vee \neg x_4) \wedge (x_3 \vee \neg x_4 \vee \neg x_5) \wedge \\ & (x_4 \vee \neg x_5 \vee \neg x_6) \wedge (x_5 \vee \neg x_6 \vee \neg x_1) \wedge (x_6 \vee \neg x_1 \vee \neg x_2) \wedge \\ & (x_1 \vee x_2 \vee x_3) \quad \wedge (x_1 \vee x_2 \vee \neg x_3) \quad \wedge (\neg x_1 \vee x_2 \vee x_3) \quad \wedge \cdot \\ & (\neg x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \end{aligned} \quad (1)$$

Φ has a unique solution: $x_1 = x_2 = \dots = x_6 = \text{true}$.

2 Materials and Methods

2.1 Design of the Library

To represent all possible variable assignments for the chosen 6-variable SAT problem, a Lipton encoding [3] was used. For each of the 6 variables x_1, x_2, \dots, x_6 , two distinct 15 base *value sequences* were designed—one representing *true* (T), X_k^T , and one representing *false* (F), X_k^F . Each of the 2^6 truth assignments was represented by a *library sequence* of 90 bases consisting of the concatenation of one value sequence for each variable. DNA molecules with library sequences are termed *library strands* and a combinatorial pool containing library strands is termed a *library*. The probes used for separating the library strands have sequences complementary to the value sequences.

Errors in the separation of the library strands are errors in the computation. Sequences must be designed to ensure that library strands have little secondary structure which might inhibit intended probe-library hybridization. The design

must also exclude sequences that might encourage unintended probe-library hybridization. To help achieve these goals, sequences were computer-generated to satisfy the following constraints:

1. Library sequences contain only A's, T's, and C's.
2. All library and probe sequences have no occurrence of 5 or more consecutive identical nucleotides; i.e. no runs of more than 4 A's, 4 T's, 4 C's or 4 G's occur in any library or probe sequences.
3. Every probe sequence has at least 4 mismatches with all 15 base alignment of any library sequence (except for with its matching value sequence).
4. Every 15 base subsequence of a library sequence has at least 4 mismatches with all 15 base alignment of itself or any other library sequence.
5. No probe sequence has a run of more than 7 matches with any 8 base alignment of any library sequence (except for with its matching value sequence).
6. No library sequence has a run of more than 7 matches with any 8 base alignment of itself or any other library sequence.
7. Every probe sequence has 4, 5, or 6 Gs in its sequence.

Constraint (1) is motivated by the assumption that library strands composed only of As, Ts, and Cs will have less secondary structure than those composed of As, Ts, Cs, and Gs [22]. Constraint (2) is motivated by two assumptions: first, that long homopolymer tracts may have unusual secondary structure and second, that the melting temperatures of probe-library hybrids will be more uniform if none of the probe-library hybrids involve long homopolymer tracts. Constraints (3) and (5) are intended to ensure that probes bind only weakly where they are not intended to bind. Constraints (4) and (6) are intended to ensure that library strands have a low affinity for themselves. Constraint (7) is intended to ensure that intended probe-library pairings have uniform melting temperatures.

The value sequences generated to represent $x_1 = F, x_2 = F, \dots, x_6 = F$ were:

$$\begin{aligned} X_1^F &= 5' - \text{TATTCTCACCATAA} - 3', & X_2^F &= 5' - \text{ACACTATCAACATCA} - 3' \\ X_3^F &= 5' - \text{CCTTTACCTCAATAA} - 3', & X_4^F &= 5' - \text{CTCCCAAATAACATT} - 3' \\ X_5^F &= 5' - \text{AACTTCACCCCTATA} - 3', & X_6^F &= 5' - \text{TCATATCAACTCCAC} - 3' \end{aligned}$$

The value sequences generated to represent $x_1 = T, x_2 = T, \dots, x_6 = T$ were:

$$\begin{aligned} X_1^T &= 5' - \text{CTATTTATATCCACC} - 3', & X_2^T &= 5' - \text{ACACCTAACTAACT} - 3' \\ X_3^T &= 5' - \text{CTACCCTATTCTACT} - 3', & X_4^T &= 5' - \text{ATCTTTAAATACCCC} - 3' \\ X_5^T &= 5' - \text{TCCATTTCTCCATAT} - 3', & X_6^T &= 5' - \text{TTTCTTCCATCACAT} - 3' \end{aligned}$$

We note that because of the nature of the constraints (which require the inspection of subsequences ≤ 15 bases long) it was only necessary to check that a special subset of all 2^6 library sequences satisfied the constraints. In particular, the sequence design program checked that the library sequences $X_1^T X_2^T X_3^T X_4^T X_5^T X_6^T$, $X_1^F X_2^F X_3^F X_4^F X_5^F X_6^F$, $X_1^F X_2^T X_3^F X_4^T X_5^F X_6^T$, and

$X_1^T X_2^F X_3^T X_4^F X_5^T X_6^F$ simultaneously satisfied constraints (1–7). These sequences contain, as subsequences, all 15 base subsequences that occur in the full 64 sequence set of library sequences. Thus, for longer library sequences, the number of constraints that need to be checked does not increase exponentially with the number of variables but rather as the square of the number of variables. We denote the reverse complements of X_k^T and X_k^F as \overline{X}_k^T and \overline{X}_k^F , respectively. We sometimes refer to “ X_k^T or X_k^F ” and “ \overline{X}_k^T or \overline{X}_k^F ” as X_k and \overline{X}_k , respectively.

2.2 Synthesis of the Library and Probes

The 6-variable library strands were synthesized by employing a mix-and-split combinatorial synthesis technique [23]. Oligonucleotide synthesis was performed on a dual column ABI 392 DNA/RNA Synthesizer (Applied Biosystems, Foster City, CA) at a 1 μ mole scale on CPG solid support. The library strands were assigned library sequences with X_1 at the 5' end and X_6 at the 3'-end (5' – $X_1 - X_2 - X_3 - X_4 - X_5 - X_6 - 3'$). Thus synthesis began by assembling the two 15 base oligonucleotides with sequences X_6^T and X_6^F in separate columns. The columns were then removed from the synthesizer and opened; the CPG beads in the columns were removed and mixed together. One half of the beads were returned to the first column and the other half to the second. Synthesis continued with sequences X_5^T and X_5^F . This process was repeated until all 6 variables had been treated.

Twelve probes, having sequences $\overline{X}_k^F, \overline{X}_k^T, k = 1 \dots 6$ and modified at the 5'-end with AcryditeTM, were obtained from Operon Technologies Inc. (Alameda, CA).

2.3 Library Capture Analysis

To determine the efficiency of library capture and release by gel-embedded probes, capture experiments were undertaken. In this experiment, a library similar to that described in [22] was used but the synthesis was performed using a polystyrene rather than a CPG support.

Preparation of gels. Capture gels were prepared in 1 mm x 10 cm x 10 cm plastic gel cassettes (Novex). The upper half of the gel cassette was divided into three parts by inserting ~1-mm thick, ~5-mm wide, plastic spacers. Approximately 7 ml of 10% acrylamide solution were poured into the cassette, enough to cover the bottom 0.5 cm of the dividing spacers, and allowed to polymerize. After the acrylamide had polymerized for 10 minutes, any unpolymerized solution was shaken off, and the top of the polymerized gel rinsed with 1X TBE buffer. Capture layers were then polymerized on top of the already polymerized gel. In each partition of the gel, 100 μ l of 10% acrylamide solution containing 15 μ M of the appropriate probe were allowed to polymerize. Again polymerization was allowed to proceed for 10 minutes, excess solution shaken off, and top of the gel layer rinsed with 1X TBE. At this point, dividing spacers were removed and top

of the gel rinsed one more time with 1X TBE. The gel was then topped off with 10% acrylamide solution and appropriate combs for well formation inserted. Two sets of capture gels were prepared and used in the library capture experiment.

Running the gels. For one set of gels, electrophoresis was carried out in the fridge (cold) at 4°C in order to observe capture. For the other set of gels, electrophoresis was carried out using a gel box with a water circulator (hot) set to 75°C. In both cases gels were put in the electrophoresis chamber and allowed to come to thermal equilibrium before commencing electrophoresis. Electrophoresis was carried out at 10 volt/cm² in all cases. It was observed that in the higher temperature electrophoresis went at a faster rate than when electrophoresis was carried out in the cold. After electrophoresis was complete, gels were dried on a gel dryer for 30 minutes at 40°C. After drying, the gels were put in Phosphor Storage Screens and exposed overnight.

2.4 Confirming Integrity of the Library via PCR

To verify the degeneracy and integrity of the library, the library was amplified via PCR. Twenty PCR reactions were performed on the library using 5'-end primers with sequences X_1^T or X_1^F and 3'-end primers with sequences $\bar{X}_2^T, \dots, \bar{X}_6^T$ or $\bar{X}_2^F, \dots, \bar{X}_6^F$.

2.5 The Algorithm

Coupling of the AcryditeTM phosphoramidite to DNA probes allows the probes to be immobilized in a polyacrylamide gel matrix. During electrophoresis at low temperatures, such probes hybridize with and capture passing DNA molecules bearing complementary subsequences. DNA molecules without complementary subsequences pass through the gel relatively unhindered. Captured DNA strands can be released by running electrophoresis at a temperature higher than the melting temperature of a probe/probe-complement hybrid. Released molecules can be used in subsequent steps as required. Using this approach, our algorithm is as follows:

1. For each of the 11 clauses of Φ prepare a polyacrylamide gel *capture layer* containing three AcryditeTM-modified probes, one for each literal in the clause. (If x_k appears in the clause, add a probe with sequence \bar{X}_k^T ; if $\neg x_k$ appears add a probe with sequence for \bar{X}_k^F .) Place the capture layers in sequence within a single gel. Place the library into the gel preceding the first capture layer.
2. Cool the area of the gel containing the first capture layer while heating the areas of the gel preceding and following it. Begin electrophoresis to move the library through the first capture layer. Molecules encoding truth assignments satisfying the first clause will be captured in the first capture layer, while molecules encoding non-satisfying assignments will run through the first capture layer and continue beyond the second capture layer.

3. Cool the area of the gel containing the second capture layer while heating the areas of the gel preceding and following it. Molecules captured in the first capture layer will be released to move through the second capture layer. Released molecules encoding truth assignments satisfying the second clause will be captured in the second capture layer, while molecules encoding non-satisfying assignments will run through the second capture layer and continue beyond the third capture layer.

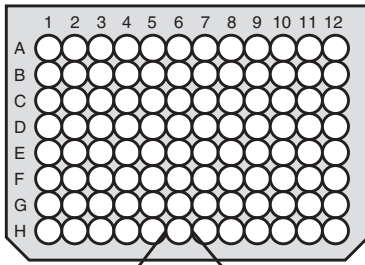
This process is repeated for each of the remaining 9 capture layers. The final (11th) capture layer will capture only those molecules which have been retained in all 11 capture layers and hence encode truth assignments satisfying each clause of Φ . These *answer strands* are extracted from the final capture layer, PCR-amplified and sequenced.

2.6 Construction and Running of the Computer

Preparation of the modules. For each clause in Φ a *clause module* (Fig. 1B) containing a 3-probe capture layer was prepared. The capture layer was prepared by mixing the three probes (chosen as described in step 1 of the algorithm above) at a concentration of 7.5 μM each in 10% acrylamide solution. 100 μl of this solution were allowed to polymerize on top of 200 μl of already-solidified 0.7% (w/v) agarose (Seakem[®] Gold, FMC BioProducts, Rockland, ME) in a well of a 96-well flat bottom plate (Nunc-Immuno Plate, Nalge Nunc, Rochester, NY). Once the acrylamide layer had polymerized (10–15 minutes), warm agarose solution was added to fill the remainder of the well and allowed to solidify. Using this method, 11 clause modules were prepared. In addition, a *library module* was prepared by mixing 500 pmols of the library with agarose and allowing the agarose to solidify in one of the wells. Other *blank modules* were prepared by allowing pure agarose to solidify in some wells.

Loading the modules. The computation was performed in a 35-cm long glass tube with an outer diameter of 0.5 cm and an inner diameter of 0.3 cm. Before loading the tube with modules, the inside of the tube was silanized with Sigmacote[®] (Aldrich, Milwaukee, WI). The tube was then loaded with modules by a squishing method (Fig. 1C–E) wherein the tube was pushed into the appropriate well. This process cut a cylindrical core of gel from the well, transferred it into the tube, and forced resident modules upward. To ensure good contact between the successive modules and to reduce the possibility of air bubble formation at the interface, prior to each squish, resident modules were pushed downward until a small bit of the lowest module protruded from the tube. First several blank modules were added to the tube, followed by alternating clause modules and blank modules. Thus each capture module was separated from the next by a blank module of pure agarose gel. After all of the 11 clause modules had been loaded in the tube, a blank module and then finally the library module were loaded.

A



B

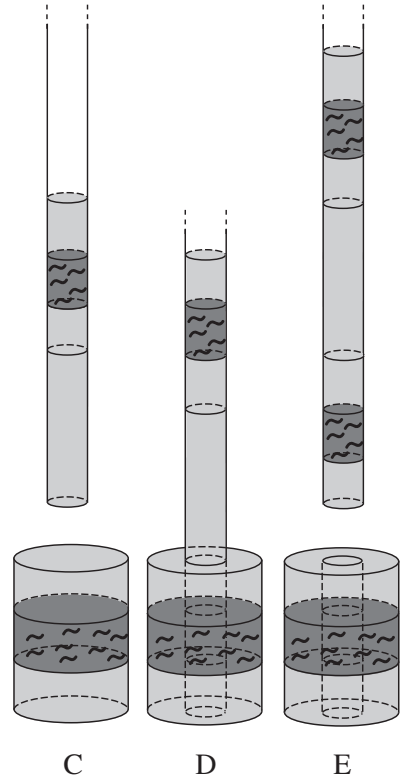
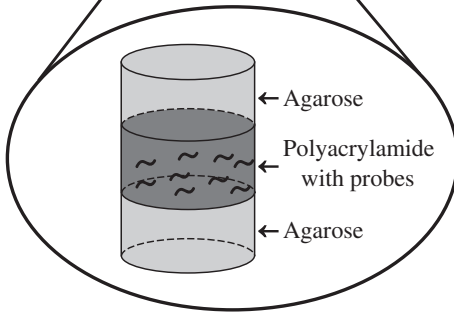


Fig. 1. Preparation of a clause module. (A) A 96-well flat bottom plate was used as a mold for the clause modules (as well as the library and blank modules). (B) To make a clause module, a polyacrylamide capture layer (with appropriate probes) was poured on top of already-solidified agarose gel. The well was then topped-off with agarose. (C–E) Loading of a clause module. (C) A glass tube holding one clause module and one blank module is positioned over a well holding a second clause module. (D) The glass tube is lowered into the well and the resident modules are pushed upwards. (E) The tube, now holding two clause modules and one blank module, is withdrawn.

Heating and cooling the capture layers. To keep the temperature high or low at a given position on the glass tube, three movable water jackets were assembled by drilling 0.5-cm holes in plastic drying tubes (Aldrich). These water jackets were connected to hot or cold water circulators with plastic tubing and slid onto the glass tube. Figure 2 shows a schematic of the final apparatus.

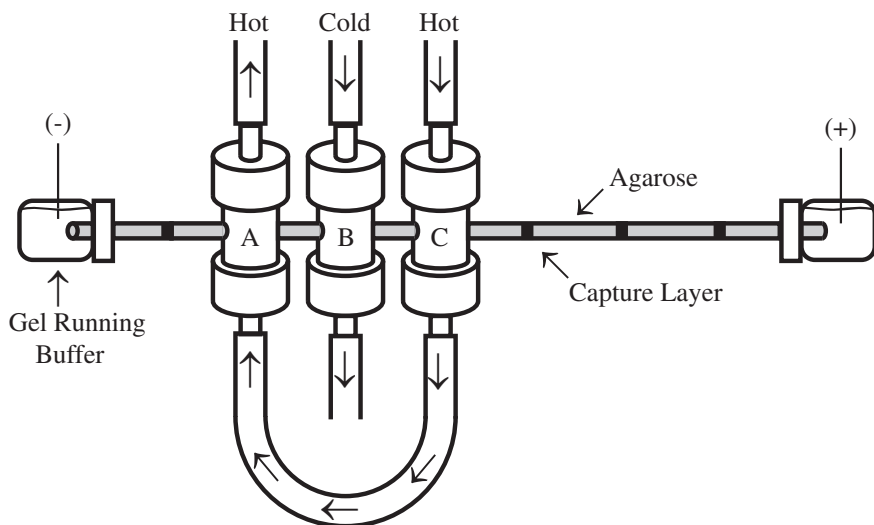


Fig. 2. Apparatus assembled for computation. A 35-cm glass tube loaded with the library module, 11 blank modules, and 11 clause modules was fitted with three water jackets (A-C). Library strands in the capture layer inside of (A) were released and moved into the capture layer inside of (B). There, library strands with subsequences complementary to the probes were captured and retained. The rest of the strands passed into the capture layer inside of (C) but because (C) was kept hot the strands passed through unhindered.

2.7 Computation

The ends of the assembled apparatus were inserted into capped glass Liquid Scintillation Vials (Wheaton, Millville, N.J.) containing, 1X TBE gel running buffer and electrodes. The water jackets were adjusted so that the cold water jacket was positioned over the first capture layer. Throughout the computation, the hot circulating water bath was set at 75°C while the cold bath was maintained at 4°C. Gel electrophoresis was performed at a constant voltage of 307 volts, ~3 mA. After 30 minutes had passed, the electrophoresis was stopped, each water jacket was moved to the next capture layer, and the electrophoresis was restarted. This process was repeated for all 11 capture layers.

Next the gel was extruded from the glass tube and the final capture layer was dissected away, crushed and soaked in 5 ml of water. The captured answer strands were extracted from the gel by incubating at 65°C for 12 hours. After extraction, the DNA was lyophilized and reconstituted into 200 μ l of water. The DNA was desalted using a Microcon 30 Microconcentrator column (Millipore, Bedford, MA) and reconstituted in 500 μ l of water.

2.8 Determination of Answer Strand

PCR. PCR amplification of the answer strands was performed on a PE Applied Biosystems GeneAmp[®] PCR System 9700 (Perkin Elmer, Foster City, CA). Five PCR reactions were run. For the first four reactions oligos with sequences X_1^T or X_1^F were used as 5'-end primers and AcryditeTM-modified probes with sequences \bar{X}_6^T or \bar{X}_6^F were used as 3'-end primers. In the fifth PCR reaction all four primers were used simultaneously. The PCR reactions were performed using $\frac{1}{5000}$ th (by volume) of the reconstituted answer strands and 10 pmols each of the appropriate primers in a final reaction volume of 50 μ l that contained 50 mM KCl, 1.5 mM $MgCl_2$, 10mM tris (pH 8.8), 200 μ M of each dNTP, and 1 unit of Taq DNA Polymerase (Promega, WI). The reaction mixture was preheated to 95°C and thermocycled (95°C 15s, 40°C, 45s, 72°C, 90s) 35 times. To determine the number of correct answer strands recovered, additional PCR reactions (using all primer pairs and all four primers) were performed using fractions of the recovered answer strands from $\frac{1}{500}$ th down to $\frac{1}{5 \times 10^{13}}$ th.

Sequencing. Two sequencing reactions were run. The product of PCR amplification using primers with sequences X_1^T and \bar{X}_6^T was sequenced using a primer with sequence X_1^T . In addition, the product of PCR amplification using all four primers (X_1^T , X_1^F , \bar{X}_6^T , and \bar{X}_6^F) was sequenced using a primer corresponding to X_1^F .

Prior to sequencing, 5 μ l of the PCR product was incubated for 15 minutes at 37°C with 1 μ l (2 units) of Shrimp Alkaline Phosphatase and 1 μ l (10 units) of Exonuclease I (PCR Product Pre-Sequencing Kit, USB, Cleveland, OH). This pretreatment was performed to destroy any dNTP's, primers and extraneous single-stranded molecules left over from the PCR reaction that might have interfered with the sequencing reaction. After incubation the reaction tube was heated to 80°C for 15 minutes to inactivate the enzymes.

Pretreated PCR product was sequenced using the Thermo Sequenase Radiolabeled Terminator Cycle Sequencing Kit (USB). The sequencing reaction was run through 30 thermocycles (denatured at 95°C for 15s, annealed at 40°C for 45s, and extended at 72°C for 90s) on a GeneMate Thermocycler (ISC Bioexpress, Kaysville, UT).

3 Results

3.1 Library Capture Analysis

Figure 3 shows the capture of the library using each of the twelve possible probes. At low temperature library was captured on each of the probes. This confirmed both that library strands with subsequences corresponding to each value sequence were present in the library and that the probes were good (sometimes incompletely modified probes failed to copolymerize into gels). At high temperature, library passed the probes unhindered suggesting that library strands could be efficiently released from probes at each step in the computation.

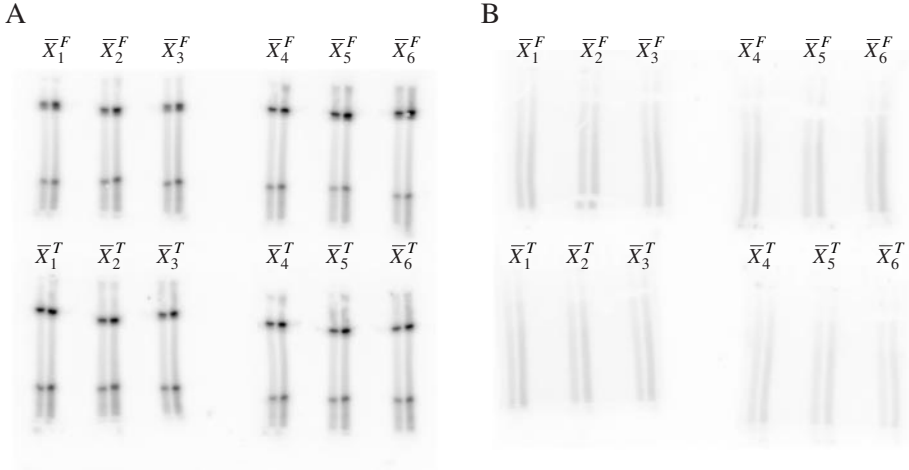


Fig. 3. Capture of the library by gel-embedded probes. (A) Twelve probes $\bar{X}_1^F, \dots, \bar{X}_6^F$ and $\bar{X}_1^T, \dots, \bar{X}_6^T$ were used to assay the capture of library strands bearing sequences X_1^F, \dots, X_6^F and X_1^T, \dots, X_6^T at low temperature. Upper bands show capture of the library on probes, lower bands show uncaptured library that presumably does not bear a subsequence complementary to the probe. (B) Repetition of the experiment at high temperature shows that library strands passed the probes unhindered (and hence could be released from a capture layer) at high temperature.

3.2 Confirming the Integrity of the Library via PCR

Figure 4 shows the results of the PCR reactions. PCR products of the expected lengths were obtained confirming that library strands with the correct subsequences corresponding to each value sequence (*true* or *false*) were present and in the expected positions in the library.

3.3 Readout of the Answer Strands by PCR

Figure 5 shows the results of the PCR amplification of $\frac{1}{5000}$ th of the answer strands using all 4 combinations of the primers. When primers with sequences X_1^T and \bar{X}_6^T were used, a 90-mer PCR product was seen. For the other 3 combinations of primers, little if any amplification was seen. Amplification of the original library was seen to give a 90-mer PCR product with each of the 4 different combinations of the primers. This indicates that the answer strands were enriched for strands encoding $x_1 = T$ and $x_6 = T$. Additional PCR reactions at other dilutions revealed that incorrect strands are present in small numbers (PCR of $\frac{1}{500}$ th of the answer strands gave positive bands for all pairs of primers) and that the correct strands are present in great numbers (PCR of $\frac{1}{5 \times 10^{11}}$ th still gave a positive band for X_1^T and \bar{X}_6^T primers). Assuming that PCR allows

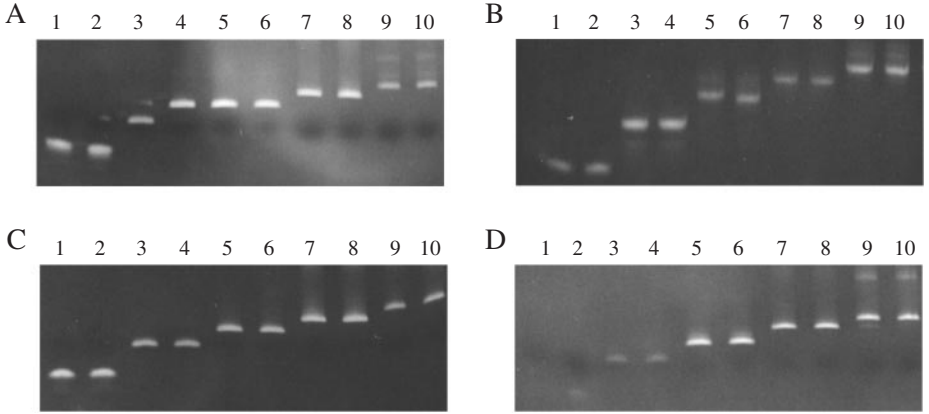


Fig. 4. PCR analysis of the original library. (A) $X_1^T, \bar{X}_2^T, \dots, \bar{X}_6^T$ probes. (B) $X_1^F, \bar{X}_2^F, \dots, \bar{X}_6^F$ probes. (C) $X_1^T, \bar{X}_2^F, \dots, \bar{X}_6^F$ probes. (D) X_2^F and $\bar{X}_2^F, \dots, \bar{X}_6^F$ probes. In each panel all lanes have X_1 as one primer and have as the other primer: lanes 2 and 3 \bar{X}_2 probe primer; lanes 3 and 4, \bar{X}_3 primer except for in panel (A) where only lane 3 has the \bar{X}_3 probe primer while lane has \bar{X}_4 probe primer; lanes 5 and 6, \bar{X}_4 primer, but see above for panel (A); lanes 7 and 8, \bar{X}_5 primer; and lanes 9 and 10, \bar{X}_6 primer.

the detection of single molecules, these PCR results allow us to approximate the number of correct strands and incorrect strands present in the recovered answer strands. Assuming that strands amplified by the primers X_1^T and \bar{X}_6^T are correct strands, at least 5×10^{11} correct strands were present in the answer strands. Given that 500 pmols (3.0×10^{14} strands) were input to the computation, $\frac{1}{64}$ th of these, or 4.7×10^{12} correct strands were input to the computation. Thus approximately 11% of the correct strands were retained at the end of the computation. Assuming that those strands amplified by the primers X_1^T and \bar{X}_6^F were a single type of incorrect strand (bearing X_6^F and X_k^T for $k = 1 \dots 5$) there were less than 5000 of such strands. Assuming that all types of incorrect strands are present with this frequency means that at most 315,000 incorrect strands were present in the answer strands. This suggests correct strands outnumbered incorrect strands by a factor of at least 1.6 million, an enrichment from their original proportions by a factor of 100 million.

3.4 Sequencing of the Answer Strands

Figure 6A and B show the results of sequencing the amplified answer strands using a primer with sequence X_1^T . Figure 6 shows that there is no degeneracy at any position, indicating that a unique computational solution was obtained. The unique solution corresponded to $x_2 = T, x_3 = T \dots, x_6 = T$. Since $x_1 = T$ and $x_6 = T$ had already been established in the PCR step, it can be concluded

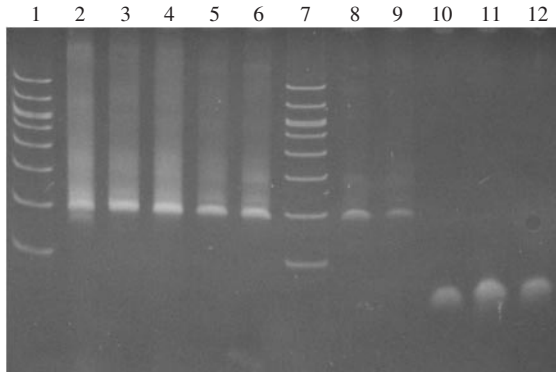


Fig. 5. Readout of the answer by PCR. Lanes 1 and 7, molecular size marker ladder. Lanes 2 – 6, library. Lanes 8 – 12, diluted answer strands. Lanes 2 and 8, all four primers. Lanes 3 and 9, X_1^T and \bar{X}_6^T probe. Lanes 4 and 10, X_1^T and \bar{X}_6^T probe. Lanes 5 and 11, X_1^F and \bar{X}_6^F probe. Lanes 6 and 10, X_1^F and \bar{X}_6^F probe.

that the answer strands correspond to $x_1 = T$, $x_2 = T$, $x_3 = T$, $x_4 = T$, $x_5 = T$, and $x_6 = T$, indicating a successful computation.

4 Prospects for Scaling Up

Whether SAT problems of greater size may be solved depends on the difficulty of scaling up each of three procedures: design of the library strands, synthesis of the library strands, and execution of the computation. As for the first procedure, we note that the sequences X_1^T, \dots, X_6^T and X_1^F, \dots, X_6^F are a subset of 72 sequences designed for a larger 36-variable SAT problem and that sequences for 50-variable SAT problems with the same constraints have been designed. Assuming that longer library strands composed of these sequences perform as well as their shorter variants, sequence design does not seem to be a limiting factor for the solution of SAT problems with up to ten times as many variables as that solved here.

As for the second procedure, we plan to synthesize library strands for a 20-variable SAT problem via the ligation of three pools of shorter library strands: two pools of 105-base long 7-variable library strands and one pool of 90-base long 6-variable library strands. The 6-variable and 7-variable library strands have already been synthesized by a mix-and-split synthesis. Each library is being tested separately by running a capture analysis and simple computation as described for the 6-variable library. We have just begun experiments to ligate these pools into a full 20-variable library.

As for the third procedure, we believe that the results of our 6-variable computation show that our ability to capture, release, and recover correct answer

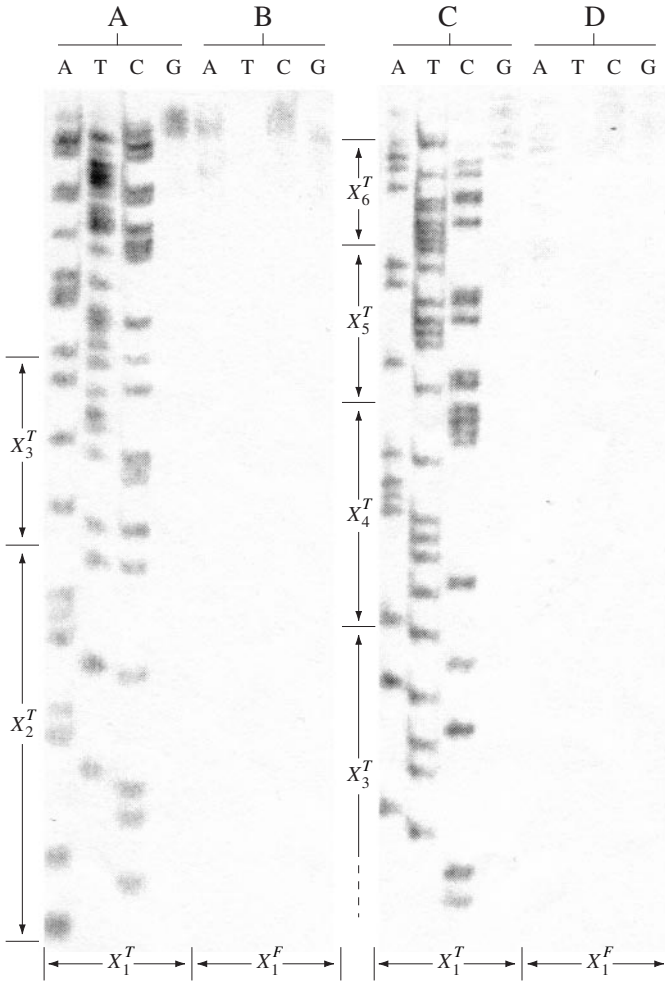


Fig. 6. Sequencing of the diluted answer strands. Termination nucleotides are shown at the top of each lane. The primer used in each sequencing reaction is indicated at the bottom of the lanes. Lanes (A) and (C) show sequencing of the PCR amplification of answer strands using primers with sequences X_1^T and X_6^T (lane 9 from Fig. 5). Lanes (A) shows that the strands contain, as subsequences, the sequences X_2^T and X_3^T while lanes (B) show the answer strands contain, as subsequences, part of X_3^T and all of X_4^T, \dots, X_6^T . Lanes (B) and (D) show sequencing of the PCR amplification of answer strands using all four primers (lane 8 from Fig. 5). The absence of sequencing product in these lanes indicates that no strands representing any assignment including $x_1 = F$ were present in the answer strands.

strands is good enough to complete a successful 20-variable computation. Consider that after capture in and release from 11 clause modules $\sim 11\%$ of the correct answer strands were recovered. This suggests that in each module approximately 82% of the correct answer strands were captured. In an analogous 20-variable computation, starting with 500 pmols of library strands, there would be roughly 300 million correct answer strands. After passing 25 clause modules (for the analogous SAT formula) approximately 2 million correct answer strands would remain—enough to be easily detected by PCR.

5 Discussion

We have carried out a successful DNA computation on a 6-variable SAT problem. The correct solution was culled from 64 alternatives. This is slightly smaller than the number of alternative solutions (512) recently handled at Princeton by Faulhammer et al. [16] and slightly more than the number (16) handled the University of Wisconsin-Madison by Liu et al. [20]. By solving small computational problems, these experimentalists and others have demonstrated the viability of several different architectures for DNA computing. It seems clear that the next objective should be the solution of problems which are beyond the capabilities of humans without the aid of electronic computers. Our success with a 6-variable 11-clause 3-SAT problem fortifies our view that we now possess the tools necessary to carry out such a computation. We are currently in the process of synthesizing a 20-bit library in order to solve a 20-variable SAT problem in the near future. We are also optimistic about the prospects of building an automated device for carrying out such computations. Despite our optimism, we must still acknowledge that the road to a DNA computer capable of solving computational problems which cannot be solved by electronic computers is a difficult one. In our opinion, creation of such a molecular computer will not be accomplished by incremental improvements in current approaches—breakthroughs will be needed.

Acknowledgements. This work was supported by grants from the National Aeronautics and Space Administration/Jet Propulsion Laboratory, the Defense Advanced Research Projects Agency, the Office of Naval Research, and the National Science Foundation.

References

1. Leonard M. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266:1021–1024, November 11, 1994.
2. Richard J. Lipton. DNA solution of hard computational problems. *Science*, 218:17–26, 1996.

3. Dan Boneh, Christopher Dunworth, and Richard J. Lipton. Breaking DES using a molecular computer. In Richard J. Lipton and Eric B. Baum, editors, *DNA Based Computers: Proceedings of a DIMACS Workshop, April 4, 1995, Princeton University*, volume 27 of *DIMACS: Series in Discrete Mathematics and Theoretical Computer Science*, pages 37–65, Providence, RI, 1996. American Mathematical Society.
4. Leonard M. Adleman, Paul W. K. Rothmund, Sam Roweis, and Erik Winfree. On applying molecular computation to the data encryption standard. *Journal of Computational Biology*, 6(1):53–63, 1999.
5. N. Jonoska and S. A. Karl. A molecular computation of the road coloring problem. In Baum and Landweber [25], pages 87–96.
6. Erik Winfree, Xiaoping Yang, and Nadrian C. Seeman. Universal computation via self-assembly of DNA: Some theory and experiments. In Baum and Landweber [25], pages 191–213.
7. Erik Winfree. Whiplash PCR for $O(1)$ computing. In *Proceedings of the 4th DIMACS Meeting on DNA Based Computers, held at the University of Pennsylvania, June 16-19, 1998*, pages 175–188.
8. Martyn Amos, Alan Gibbons, and David Hodgson. Error-resistant implementation of DNA computations. In Baum and Landweber [25], pages 151–161.
9. Michail G. Lagoudakis and Thomas H. LaBean. 2-D DNA self-assembly for satisfiability. In *Proceedings of the 5th DIMACS Meeting on DNA Based Computers, held at the Massachusetts Institute of Technology, June 14-15, 1999*, pages 139–152.
10. F. Guarnieri, M. Fliss, and Carter Bancroft. Making DNA add. *Science*, 273(5272):220–223, 1996.
11. Q. Ouyang, P. D. Kaplan, L. Shumao, and A. Libchaber. DNA solution of the maximal clique problem. *Science*, 278:446–449, 1997.
12. Nobuhiko Morimoto, Masanori Arita, and Akira Suyama. Solid phase DNA solution to the Hamiltonian Path Problem. In Rubin and Wood [24], pages 83–101.
13. Thomas H. Leete, Joshua P. Klein, and Harvey Rubin. Bit operations using a DNA template. In Rubin and Wood [24], pages 159–171.
14. Kensaku Sakamoto, Daisuke Kiga, Ken Komiya, Hidetaka Gouzu, Shigeyuki Yokoyama, Shuji Ikeda, Hiroshi Sugiyama, and Masami Hagiya. State transitions by molecules. In *Proceedings of the 4th DIMACS Meeting on DNA Based Computers, held at the University of Pennsylvania, June 16-19, 1998*, pages 87–99.
15. Julia Khodor and David K. Gifford. Design and implementation of computational systems based on programmed mutagenesis. In *Proceedings of the 4th DIMACS Meeting on DNA Based Computers, held at the University of Pennsylvania, June 16-19, 1998*, pages 101–107.
16. Dirk Faulhammer, A. R. Cukras, Richard J. Lipton, and Laura F. Landweber. When the knight falls: On constructing an RNA computer. In *Proceedings of the 5th DIMACS Meeting on DNA Based Computers, held at the Massachusetts Institute of Technology, June 14-15, 1999*, pages 1–7.
17. Junghei Chen, Eugene Antipov, Bertrand Lemieux, Walter Cedeño, and David Harlan Wood. *In vitro* selection for a Max 1s DNA genetic algorithm. In *Proceedings of the 5th DIMACS Meeting on DNA Based Computers, held at the Massachusetts Institute of Technology, June 14-15, 1999*, pages 23–37.
18. H. Yoshida and A. Suyama. Solution to 3-SAT by breadth first search. In *Proceedings of the 5th DIMACS Meeting on DNA Based Computers, held at the Massachusetts Institute of Technology, June 14-15, 1999*, pages 9–20.

19. Masahito Yamamoto, Jin Yamashita, Toshikazu Shiba, Takuo Hirayama, Shigeharu Takiya, Keiji Suzuki, Masanabu Munekata, and Azuma Ohuchi. A study on the hybridization process in DNA computing. In *Proceedings of the 5th DIMACS Meeting on DNA Based Computers, held at the Massachusetts Institute of Technology, June 14-15, 1999*, pages 99–108.
20. 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.
21. Sam Roweis, Erik Winfree, Richard Burgoyne, Nickolas V. Chelyapov, Myron F. Goodman, Leonard M. Adleman, and Paul W. K. Rothemund. A sticker-based model for DNA computation. *Journal of Computational Biology*, 5(4):615–629, 1998.
22. Kalim Mir. A restricted genetic alphabet for DNA computing. In Baum and Landweber [25], pages 243–246.
23. A. R. Cukras, Dirk Faulhammer, Richard J. Lipton, and Laura F. Landweber. Chess games: A model for RNA-based computation. In *Proceedings of the 4th DIMACS Meeting on DNA Based Computers, held at the University of Pennsylvania, June 16-19, 1998*, pages 27–37.
24. Harvey Rubin and David Harlan Wood, editors. *DNA Based Computers III: DIMACS Workshop, June 23-25, 1997*, volume 48 of *DIMACS: Series in Discrete Mathematics and Theoretical Computer Science.*, Providence, RI, 1999. American Mathematical Society.
25. Eric B. Baum and Laura F. Landweber, editors. *DNA Based Computers II: DIMACS Workshop, June 10-12, 1996*, volume 44 of *DIMACS: Series in Discrete Mathematics and Theoretical Computer Science.*, Providence, RI, 1998. American Mathematical Society.

Diophantine Equations and Splicing: A New Demonstration of the Generative Capability of H Systems

Pierluigi Frisco

L.I.A.C.S., Leiden University,
Niels Bohweg 1, 2333 CA Leiden, The Netherlands
`pier@liacs.nl`

Abstract. Systems based on the splicing operation are computationally complete. Usually demonstrations of this are based on simulations of type-0 grammars. We propose a different way to reach this result by solving Diophantine equations using extended H system with permitting context. Completeness then follows from Matiyasevich's theorem stating that the class of Diophantine sets is identical to the class of recursive enumerable sets.

Solutions to a Diophantine equation are found in parallel. The numbers are coded in base one.

1 Introduction

Recently, many different formal models of computation have been proposed under the inspiration of biological processes. DNA based computation, generally speaking, considers the transformation of biological molecules as computational steps. Splicing systems (see [5], [9]) are a generative mechanism based on the splicing operation as a model of DNA recombination. If these systems have finite sets of axioms and rules defining splicing they generate regular languages (see details in [12]). Keeping finite both sets the generative power can only be increased by introducing certain control systems. In this way splicing systems can generate recursive enumerable languages.

For all proposed control systems the demonstration of this universality is based on the simulation of type-0 grammars (see [3] and [1] for permitting and forbidding context; [10] for target languages, [11] for programmed and evolving H systems, [7] for double splicing, [2] and [3] for multisets).

Here we obtain the same result by a totally different approach. We generate a recursive enumerable set of vectors using the correspondence of the class of Diophantine sets with the one of Turing semi-decidable sets defined by Matiyasevich theorem described in [6].

An extended H system with permitting context solving in parallel Diophantine equations is described in Section 4. Its operation corresponds to the parallel execution of a Turing machine on all its possible inputs. The H system, called *solver*, receives as input a polynomial $P(\bar{x})$ and, in parallel, computes the value

of P for all possible non-negative integer values of its variables. Final strings indicate the value of the unknowns for solutions of the polynomial equation $P(\bar{x}) = 0$.

Slightly modifying the (final alphabet of the) system it is possible to generate all strings indicating the value of the polynomial for certain values of the unknowns; in this case the system is a *generator* of pairs $(\bar{x}, P(\bar{x}))$. The specific solutions of the equation $P(\bar{x}) = N$ can be found by another H system, called *filter*, receiving a specific number N as input and selecting, from all the strings created by the generator, the ones giving N as result.

This last strategy (described in Section 5) corresponds to what in [8] is called *computing by carving*.

2 An Overview on Diophantine Equations

We give definitions strictly related with our work; more general information may be found in [6].

A *Diophantine equation* is an equation of the form:

$$D(x_1, \dots, x_n) = 0, \quad (1)$$

where D is a polynomial with integer coefficients. An equation as (1) can express a system of k equations. This is because the set of solutions of the system

$$\begin{aligned} D_1(x_1, \dots, x_n) &= 0 \\ &\vdots \\ D_k(x_1, \dots, x_n) &= 0 \end{aligned}$$

coincides with the set of solutions of the single equation

$$D_1^2(x_1, \dots, x_n) + \dots + D_k^2(x_1, \dots, x_n) = 0$$

Given a Diophantine equation it is possible to reduce the problem of finding whether it has an integer solution to the one of deciding whether it has non-negative solution and vice-versa. To see this let us consider an equation as (1) and the system

$$\begin{aligned} D(x_1, \dots, x_n) &= 0 \\ x_1 &= y_{1,1}^2 + y_{1,2}^2 + y_{1,3}^2 + y_{1,4}^2 \\ &\vdots \\ x_n &= y_{n,1}^2 + y_{n,2}^2 + y_{n,3}^2 + y_{n,4}^2 \end{aligned} \quad (2)$$

Any solution in arbitrary integers of the system (3) includes a solution of (1) in non-negative integers. The converse is also true: any solution of (1) in non-negative integers x_1, \dots, x_n can be expressed by integer values of $y_{1,1}, \dots, y_{n,4}$ in (3). This relies on Lagrange's theorem stating that every non-negative integer

is the sum of four squares of integers. As said before, the system (3) can be expressed by a single equation

$$E(x_1, \dots, x_n, y_{1,1}, \dots, y_{n,4})$$

which is solvable in integers if and only if the original equation is solvable in non-negative integers.

A *family* of Diophantine equations is a relation of the form

$$D(a_1, \dots, a_m, x_1, \dots, x_n) = 0 \quad (3)$$

where D is a polynomial with non-negative integer variables $a_1, \dots, a_m, x_1, \dots, x_n$. a_1, \dots, a_m are called *parameters*, while x_1, \dots, x_n *unknowns*. Fixing values of the parameters results in the particular Diophantine equations that comprise the family. A family of Diophantine equations should not be confused with an infinite system. In the family we consider separately each polynomial obtained by fixing the parameters.

By varying the values of the parameters it is possible to obtain, in the same family, equations that have solutions and others that do not. The family of Diophantine equation (3) defines a set \mathfrak{M} composed by the m -tuples of values of the parameters for which there is a solution of (3)

$$\langle a_1, \dots, a_m \rangle \in \mathfrak{M} \iff \exists x_1 \dots x_n [D(a_1, \dots, a_m, x_1, \dots, x_n) = 0].$$

A set defined in this way is called *Diophantine*.

Hilbert's tenth problem asks a universal method (i.e., an algorithm) for deciding the solvability of a Diophantine equation with integer coefficients. Yuri Matiyasevich (see [6]) demonstrated that the class of Diophantine sets is identical to the class of Turing recursively enumerable (RE) sets of integer tuples. This implies that the existence of a solution cannot be decided. However, considering that a RE set of vectors can be enumerated by a finitely described procedure, our algorithm executes this procedure in parallel generating non-negative integer solutions to a Diophantine equation.

3 An Overview on Splicing

We give definitions strictly related with our work; more general information may be found in [12]. Consider an alphabet V and two special symbols, $\#$ and $\$$ not in V . A *splicing rule* is a string of the form $r = u_1 \# u_2 \$ u_3 \# u_4$, where $u_1, u_2, u_3, u_4 \in V^*$. For such a splicing rule r and strings $x, y, z, w \in V^*$ we write:

$$\begin{aligned} (x, y) \vdash_r (z, w) \text{ iff } & x = x_1 u_1 u_2 x_2, y = y_1 u_3 u_4 y_2, \\ & z = x_1 u_1 u_4 y_2, w = y_1 u_3 u_2 x_2, \\ & \text{for some } x_1, x_2, y_1, y_2 \in V^*, \end{aligned}$$

indicating that x and y splice according to r giving z and w .

Based on this operation the notion of an *H scheme* can be defined as pairs $\sigma = (V, R)$ where V is an alphabet and $R \subseteq V^* \# V^* \$ V^* \# V^*$ is a set of splicing rules. For an H scheme and a language $L \subseteq V^*$ we define

$$\begin{aligned} \sigma(L) &= \{z \in V^* \mid (x, y) \vdash_r (z, w) \text{ or } (x, y) \vdash_r (w, z), \\ &\quad \text{for some } x, y \in L, r \in R, w \in V^*\}, \\ \sigma^0(L) &= L, \\ \sigma^{i+1}(L) &= \sigma^i(L) \cup \sigma(\sigma^i(L)), \quad i \geq 0, \text{ and} \\ \sigma^*(L) &= \bigcup_{i \geq 0} \sigma^i(L). \end{aligned}$$

If we consider two families of languages FL_1 and FL_2 , we define:

$$H(FL_1, FL_2) = \{\sigma^*(L) \mid L \in FL_1 \text{ and } \sigma = (V, R), R \in FL_2\}.$$

We denote by FIN, REG the families of finite and of regular languages respectively. We have (see details in [12])

$$FIN \subset H(FIN, FIN) \subset REG.$$

An *extended H system* is a construct $\gamma = (V, T, A, R)$, where V and T are alphabets such that $T \subseteq V$ (T is called *terminal* alphabet), A is a language on V (A is the set of *axioms*), and R is a set of splicing *rules* over V . The language generated by γ is $L(\gamma) = \sigma^*(A) \cap T^*$, where σ is the H scheme (V, R) .

If both sets A and R are finite, then $L(\gamma)$ is regular.

If we want an H system having A and R finite generating more than regular languages, then a control has to be added to the splicing operation. There are many classes of controlled systems and most of them characterize the family of recursively enumerable languages (see [3] and [12]).

The type of control we choose in order to implement the solver uses *permitting contexts* this means that rules may only be applied when given strings (the context) are present. Formally an extended H system with *permitting contexts* is a quadruple $\gamma = (V, T, A, R)$, where V is an alphabet, $T \subseteq V$, A is a finite language over V , and R is a finite set of triples $p = (r; C_1, C_2)$, with $r = u_1 \# u_2 \$ u_3 \# u_4$ being a splicing rule over V and C_1, C_2 being finite subsets of V^* . For $x, y, z, w \in V^*$ and $p \in R, p = (r; C_1, C_2)$, we define $(x, y) \vdash_p (z, w)$ iff $(x, y) \vdash_r (z, w)$, every element of C_1 appears as a substring in x and every element of C_2 appears as a substring in y ; when $C_1 = \emptyset$ and/or $C_2 = \emptyset$, then no condition on x , and/or y , is imposed.

To understand better the work of H systems used in the subsequent section of this paper, all triples are numbered.

4 Solver

Considering what described is Section 2 it is easy to understand that the creation of a system based on splicing capable to find all non-negative integer solutions of

a Diophantine equation is enough to demonstrate that it can generate the class of RE set of vectors. In this section we describe an extended H system γ_D with permitting contexts which receives as input a polynomial $P(\bar{x})$ and generates strings defining the solutions of the polynomial equation $P(\bar{x}) = 0$.

As we will see most part of the work of γ_D will be done at the ends of the strings. These ends have normally the form h_β, t_β (h to the left means *head* and t to the right means *tail*) where β defines the *state* of the string. Symbols s present in strings, together with h and t (all with subscript) are used to define permitting conditions.

Moreover to *rotate* a string will mean to move symbols between h and t (with whatever subscript) from left to right or from right to left. In this context the symbols s are important to keep track of the begin and the end of the rotated substring. For instance the clockwise rotation of two symbols of the string $hs_bv_1v_2v_3s_e t$ will bring it to $hv_3s_e s_bv_1v_2t$. Note how, even if rotated, the "substring" $v_1v_2v_3$ from s_b to s_e is unchanged.

Basic operation. Rotation of a string (together with substitution, deletion or insertion of one or two rotated symbols) is the basic step in the system we describe below. Each of these operations is performed using four triples. Here we give an example for anti-clockwise rotation with substitution of an element. The generalization to the other cases is easy and left to the reader.

Let us consider the string $h_1abs_1ct_1$ which we want to rotate anti-clockwise by one symbol, changing a into A . In this way we obtain $h_1bs_1cAt_1$. The splicing uses the string h_AAt_A present as an axiom in the system.

The operation is performed by four splicing operations, the vertical bar ($|$) indicates where the splicing occurs:

$$\begin{array}{ccccccc} h_1a & | & bs_1ct_1 & & h_Abs_1c & | & t_1 & & h_A & | & bs_1cAt_A & & h_1bs_1cA & | & t_A & & h_1bs_1cAt_1 \\ h_A & | & At_A & \vdash_1 & h_1a & | & At_A & \vdash_2 & h_1 & | & at_1 & \vdash_3 & h_Aa & | & t_1 & \vdash_4 & h_Aat_A \end{array}$$

The triple 1 = $(h_1a\#\$h_A\#At_A; \{s_1\}, \emptyset)$ is used and h_1a is replaced by h_A . Now the string containing s_1 has also h_A and t_1 , as a sign that an a has been removed from the left side. The subscript of h is changed into A just to keep memory of what was just done.

After this the triple 2 = $(\#t_1\$h_1a\#At_A; \{h_A, s_1\}, \emptyset)$ changes t_1 into At_A in the string containing h_A and s_1 .

At this point the string with s_1 has completed the operation of rotating/substituting but it is in state A . Two more triples 3 = $(h_A\#\$h_1\#at_1; \{s_1, t_A\}, \emptyset)$ and 4 = $(\#t_A\$h_Aa\#t_1; \{h_1, s_1\}, \emptyset)$ change its state back into 1 so to obtain what we want.

Note that this sequence of splicing is purely consecutive: both strings obtained in one splicing step are used in the next one. This reminds us of the double splicing discussed in [7].

In the strings used by the system, numbers are expressed in base one, upper-scripts indicate the repetition of symbols.

We do not describe the system γ_D in all details but we give a comprehensible explanation of the way it works through its basic steps. This is done using

an example: the computation of the polynomial $x^2y + 3y^2z - z^2$ with values $x = 2, y = 1, z = 4$.

Let us consider $\Delta = \{x, y, z\}$ the set of unknowns in the polynomial.

The polynomial in the example will be represented in γ_D by the axiom $h_1s_b + d\tilde{x}^2\tilde{y} + d^3\tilde{y}^2\tilde{z} - d\tilde{z}^2s_et_1$. The symbols s_b and s_e indicate respectively the begin and the end of the polynomial; $\tilde{\alpha}, \alpha \in \Delta$ indicate the unknowns; $+$ and $-$ indicate sum and subtraction respectively and d 's are used to code coefficients.

The system evolves through three logical parts:

creation. Strings defining values for unknowns in the polynomial are generated.

These have the form $h_{\tilde{\alpha}}\hat{\alpha}a^nt, n \geq 0, \alpha \in \Delta$ and a 's represent the value of α coded in base one.

union. The strings defining the polynomial are joined with the ones created in the previous phase, one for each unknown. Strings obtained in this phase are called *working strings*.

computation. The polynomial is computed using values present in the working string.

Creation. The symbols $h_{\tilde{\alpha}}\hat{\alpha}t, \alpha \in \Delta$ and h_at are axioms. They can be spliced by $(\#\$h_a\#a; \{h_{\tilde{\alpha}}\}, \emptyset)$ so to obtain $h_{\tilde{\alpha}}\hat{\alpha}at$ ($\alpha \in \Delta$). The same triple can be iteratively applied to the result of the previous splicing and other copies of the axiom h_at so to create $h_{\tilde{\alpha}}\hat{\alpha}a^nt$. In this way a value (n) is associated to an unknown (α).

Union. The strings representing the polynomial are joined with the values of its unknowns following a fixed order driven by permitting conditions. As an example for $\Delta = \{x, y, z\}$ a string containing s_b and h_1 can join a string containing $h_{\tilde{z}}$ to assign a value to z . The triple $(h_1\#\$\#t; \{s_b\}, \{h_{\tilde{z}}\})$ can splice $h_1s_b + d\tilde{x}^2\tilde{y} + d^3\tilde{y}^2\tilde{z} - d\tilde{z}^2s_et_1$ with $h_{\tilde{z}}\hat{z}a^4t$ so to obtain $h_{\tilde{z}}\hat{z}a^4s_b + d\tilde{x}^2\tilde{y} + d^3\tilde{y}^2\tilde{z} - d\tilde{z}^2s_et_1$. It is easy to imagine that this operation can be repeated on this last string (using the triples $(h_{\tilde{z}}\#\$\#t; \{s_b\}, \{h_{\tilde{y}}\})$ and $(h_{\tilde{y}}\#\$\#t; \{s_b\}, \{h_{\tilde{x}}\})$) so to add the value for the other unknowns (y and x respectively) present in the polynomial until $h_x\hat{x}a^2\hat{y}a\hat{z}a^4s_b + d\tilde{x}^2\tilde{y} + d^3\tilde{y}^2\tilde{z} - d\tilde{z}^2s_et_1$ is obtained. When this happens the state of the working string is changed into 2 using two additional triples.

As the association of values to unknowns and their union with the polynomial are made in a non-deterministic way we expect all possible combinations of variables to be present in the system.

Computation. For each working string created the system will compute the value of the polynomial for the values associated to the unknowns. This operation is composed of three phases: *substitution*, *multiplication* and *addition/subtraction*.

Substitution. Reading the polynomial from s_b to s_e the first occurrence of a variable of the first monomial is substituted with c 's representing its value.

Any remaining occurrence of variables in the polynomial are not substituted at this moment.

The substitution of an unknown with its value is performed rotating anti-clockwise the working string in state 2 until the first unknown, in our example \hat{x} , is close to the head (so that $h_2\hat{x}$ is present). In the example we obtain $h_2\hat{x}^2\tilde{y} + d^3\tilde{y}^2\tilde{z} - d\tilde{z}^2s_e\hat{x}a^2\hat{y}a\hat{z}a^4s_b + dt_2$. At this point the unknown is removed and a marker M is inserted at its place, the state is changed into x (as we have to copy the value of x) and the string is rotated clockwise until $\hat{x}t_x$ is present ($h_xa^2\hat{y}a\hat{z}a^4s_b + d\tilde{x}\tilde{y} + d^3\tilde{y}^2\tilde{z} - d\tilde{z}^2s_e\hat{x}t_x$ in the example). One per time each a present on the left of the working string is copied before the marker M . Each a is marked (to keep memory that it has been copied) and, rotating the string anti-clockwise, a symbol c is added in the polynomial.

At the end of the operation all markers are removed, to recreate the previous situation, and the state is changed into 3. If the value associated to a variable is 0 (no a is present on the left side) the state of the working string is changed and the monomial removed.

In our example the substitution of the first variable in the first monomial brings to $h_3\hat{x}a^2\hat{y}a\hat{z}a^4s_b + dc^2\tilde{x}\tilde{y} + d^3\tilde{y}^2\tilde{z} - d\tilde{z}^2s_e t_3$.

Multiplication. In this phase the value of a just substituted variable is multiplied by the coefficient of the monomial so to obtain a new coefficient for a "reduced" monomial. Considering the first monomial in our example we pass from $+d\tilde{x}^2\tilde{y}$ to $+dc^2\tilde{x}\tilde{y}$ with a substitution, and then to $+d^2\tilde{x}\tilde{y}$ with a multiplication. In general in a string in state 3 a multiplication is performed: the number represented by d 's is multiplied by the number represented by c 's. This is done by subsequent additions: for each c in the monomial we insert as many i 's as d 's are present.

The implementation of the multiplication is explained using as example the string $h_As_1d^3c^3t_A$.

state A : The string is rotated clockwise until to have h_Ac and dt_A (this is bound to happen as dc is present in the string). Then the leftmost c is removed and the state is changed into B .

When h_As_1 and dt_A are present we have dealt with every c . The state is changed into C .

state B : Each d present on the right is removed and for each d removed id is added to the left, i 's are simply moved from right to left.

When s_1t_B is present the state changes into A .

state C : Rotating the string all d 's are removed and i 's are changed into d 's.

For our example rotating clockwise $h_As_1d^3c^3t_A$ we obtain $h_Ac^3s_1d^3t_A$. The two conditions indicated for state A hold so this string is spliced so to have $h_Bc^2s_1d^3t_B$. Now for each d on the right id is added to the left so to obtain $h_B(id)^3c^2s_1t_B$. Then the state is changed into A and the string is rotated anti-clockwise until $h_Ac^2s_1(id)^3t_A$. A c has been removed and 3 i 's have been added: one addition step of the multiplication has been performed.

As $h_A c$ and dt_A are present the leftmost c is removed and the state is changed into B so to have $h_B c s_1 (id)^3 t_B$. Now, rotating anti-clockwise the string, each d present on the right is removed and for each d removed id is added to the left. The symbols i 's are simply moved from right to left. Doing this the number of d 's remains always the same while i 's increase.

The repetition of these operations brings to $h_A s_1 (i^3 d)^3 t_A$, at this point the state is changed into C .

If dt_C is present in a string it is changed with t_C , in this way a d is removed; if it_C is present in a string the rightmost i is removed and a d is added to the left side of the string. In this way the string $h_C d^9 s_1 t_C$ is obtained and d 's indicate the result of the multiplication.

After a multiplication the state of a working string is changed back into 2 and the process of substitution continues: if there is another element in the same monomial it is substituted with c 's and the multiplication is performed, otherwise the substitution continues with the first occurrence of variable of the next monomial (if present).

The implementation of all substitutions and multiplications to the working string defining the polynomial will bring to $h_3 \hat{x} a^2 \hat{y} a \hat{z} a^4 s_b + d^4 + d^{12} - d^{16} s_e t_3$.

Addition/subtraction. This last part is implemented in a simple way: removing $+$ or $-$ between two monomials with the same sign or performing the subtraction between two monomials with different sign. When $h_2 s_e$ is present in a working string all monomials have been computed. The string changes state into 4 and is rotated until $h_4 s_b$ is present (in the example we have $h_4 s_b + d^4 + d^{12} - d^{16} s_e \hat{x} a^2 \hat{y} a \hat{z} a^4 t_4$). Then s_b is moved to the right and the state is changed according with the sign of the first monomial. The symbol indicating the sign is also moved to the right. In our case the sign is positive so the state is changed into $+$ ($h_+ d^4 + d^{12} - d^{16} s_e \hat{x} a^2 \hat{y} a \hat{z} a^4 s_b + t_+$ in the example). Now working strings are rotated anti-clockwise until $h_+ +$ or $h_+ -$ are present.

When $h_+ +$ is present two monomials with the same sign have to be added: the leftmost $+$ is removed and the working string continues to be rotated. Similarly the leftmost $-$ is removed from $h_- -$. In the example we pass from the string $h_+ + d^{12} - d^{16} s_e \hat{x} a^2 \hat{y} a \hat{z} a^4 s_b + d^4 t_+$ to $h_+ d^{12} - d^{16} s_e \hat{x} a^2 \hat{y} a \hat{z} a^4 s_b + d^4 t_+$.

When $h_+ -$ or $h_- +$ are present two monomials with opposite sign have to be added (this means that a subtraction has to be performed). One element is on the right of the string, the other on the left. If $h_+ -$ is present in a working string the leftmost $-$ is removed and the state changes into 5-; if $h_- +$ is present in a working string the leftmost $+$ is removed and the state changes into 5+. As in our example we have $h_+ - d^{16} \hat{x} a^2 \hat{y} a \hat{z} a^4 s_b + d^{16} t_+$ the string $h_{5-} d^{16} \hat{x} a^2 \hat{y} a \hat{z} a^4 s_b + d^{16} t_{5-}$ is obtained. At this point one element from the left and one from the right are removed. When $h_{5p} q$, $p, q \in \{+, -\}$ is present q is removed and the state is changed into h_4 ; when $h_{5p} d$ or $h_{5p} s_e$, $p \in \{+, -\}$ is present the state is changed into 4 and p is added after h_4 . A string in state 4 is rotated until we have $h_4 s_b$ so to restart addition and subtraction from the begin of the polynomial.

If $s_b + s_e$ or $s_b - s_e$ are present in a string in state 3 it means that 0 has been obtained as final value. In this case the state is changed into 6 and the string is rotated until $h_6 \hat{x}$ is present. Ends are changed into h and t .

In our example we obtain $h\hat{x}a^2\hat{y}a\hat{z}a^4s_b - s_et$, i.e., the polynomial has value 0 for the given values of the variables.

As we choose the terminal alphabet $T = \{a, \hat{a}, h, s_b, s_e, t, +, - \mid \alpha \in \Delta\}$ the obtained string is final and codes a solution of the polynomial equation $P(\bar{x}) = 0$.

5 Final Remarks

Simply changing the final alphabet of γ_D it is possible to generalize it from a solver of the polynomial equation $P(\bar{x}) = 0$ to a system computing the value of the polynomial for all non-negative integer values of its variables.

The output language created by such a system, called *generator*, can be given as input to another one, called *filter*, receiving also a specific number N as input and selecting, from all the string created by the generator, the ones giving that number as result. Figure 1 describes this idea.

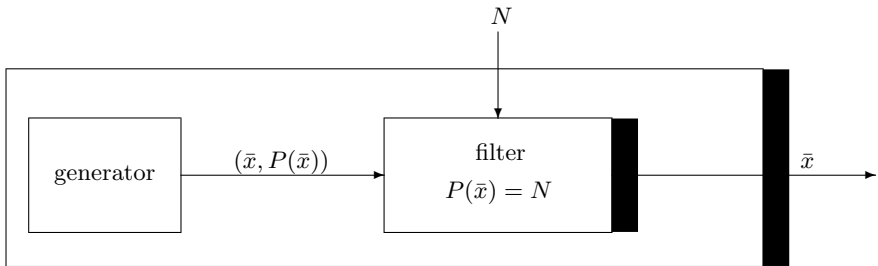


Fig. 1. Generator + filter

This mode of operation, introduced in [8], where was called *computing by carving* and implemented in [4], is not very usual in computer science or in computability theory: to produce a "complete data pool" and filter it so that only a particular kind of strings (solution) remains.

In this way parallelism is used at a higher level. Usually a problem is divided into smaller problems solved in parallel way; here a whole class of instances of a problem (to compute a polynomial for all values of its variables) is solved in parallel.

It is possible to apply this approach to a wider spectrum of problems. For instance, we may consider to have a generator of solutions in a certain interval and then, using the filter, to find the solution as close as possible to our needs.

Even if similar the idea of ‘filtering’ should not be confused with the one applied in a laboratory where beads or single strands of DNA are used to fishing out target molecules from a tube. In this case, the molecules are selected by considering a pattern present in it. Our filtering is a computing step.

Acknowledgments. I thank the Università degli Studi di Milano for its financial support to my PhD, the Universiteit Leiden, personified by Prof. G. Rozenberg, accepting me as PhD student in his friendly group of research.

A special thank Gheorghe Păun for his stimulating ideas and to Hendrik Jan Hooeboom who read the draft copy of this paper and whose suggestions were really valuable for the final one.

References

1. E. Csuhaj-Varju, R. Freund, L. Kari, Gh. Păun: DNA computing based on splicing: universality results. Proc. First Annual Pacific. Symposium in Biocomputing, Hawaii, 1996 (L. Hunter, T. E. Klein, eds.), World Scientific, Singapore, 1996, 179 - 190
2. K. L. Denninghoff, R. W. Gatterdam: On the undecidability of splicing systems. *International Journal of Computer and Mathematics*, 27 (1989), 133 - 145
3. R. Freund, L. Kari, G. Păun: DNA computing based on splicing. The existence of universal computers. Technical Report 185-2/FR-2/95, Technical Univ. Wien, 1995, and *Theories of Computer Sci.* in press
4. P. Frisco: Parallel Arithmetic with Splicing. To appear in *Romanian Journal of Information Science and Technology* (ROMJIST)
5. T. Head, Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviors, *Bulletin of Mathematical Biology*, 49 (1987), 737-759
6. Yuri Matiyasevich: *Hilbert's tenth problem*. MIT Press Cambridge, London, 1993
7. Gh. Păun: DNA computing based on splicing: universality results. Proceedings of the Second Internal Colloquium on Universal Machines and Computations, Metz, 1998, Vol. I, 67 - 91
8. Gh. Păun, (DNA) Computing by carving, *Soft Computing*, 3, 1 (1999), 30 - 36
9. Gh. Păun, On the splicing operation, *Discrete Appl. Math.*, 70 (1996), 57-79
10. Gh. Păun: Splicing systems with targets are computationally complete. Inform. Processing Letters, 59 (1996), 129 - 133
11. Gh. Păun, G. Rozenberg, A. Salomaa: Computing by splicing. Programmed and evolving splicing systems. IEEE Inter. Conf. on Evolutionary Computing, Indianapolis, 1997, 273 - 277
12. Gh. Păun, G. Rozenberg, A. Salomaa: *DNA Computing*. Springer-Verlag, 1998

About Time-Varying Distributed H Systems

Maurice Margenstern¹ and Yurii Rogozhin²

¹ Institut Universitaire de Technologie,
Université de Metz, Metz, France
LITA, EA 3097

(Laboratoire d'Informatique Théorique et Appliquée)

`margens@antares.iut.univ-metz.fr`

² Institute of Mathematics and Computer Science of the
Academy of Sciences of Moldova

`rogozhin@math.md`

Abstract. A time-varying distributed H system (TVDH system) is a splicing system which has the following feature: at different moments one uses different sets of splicing rules (these sets are called components of TVDH system). The number of components is called the degree of the TVDH system. The passing from a component to another one is specified in a cycle. It was proved by both authors (1999) that TVDH systems of degree 2 generate all recursively enumerable languages. It was made by modelling Turing machines and, in that modelling, every language is generated "step by step" or "word by word". This solution is not a fully parallel one. A.Paun (1999) presented a complete parallel solution for TVDH systems of degree 4 by modelling *type 0* formal grammars. Now we improved A.Paun's result by reducing the number of components of such TVDH systems down to 3. This question is open for 2 components, i.e. is it possible to construct TVDH systems of degree 2 which completely uses the parallel nature of molecular computations based on splicing operations (say model *type 0* formal grammars)?

We consider also original G.Paun's definition of TVDH systems and suggest a slightly different definition of TVDH systems based on the definition of H systems – extended time-varying distributed H systems (ETVDH systems). For this new definition we proved that ETVDH systems with one component generate exactly the set of all regular languages and that with two components, they generate exactly the set of all recursively enumerable languages.

1 Introduction

Starting from [1], a grounding paper on splicing computations, a lot of studies were devoted to various extensions of H systems originating from [3]. In particular, a lot of them point at the possible universality power of such systems. Extending the original definition, paper [11] defined the notion of *test tube* and proved the universality of an extended H system with a finite number of test tubes. But no indication on the number of test tubes needed in order to obtain

universality was given in [11]. That number was first set to 9 [2], then to 6 [12], and finally established to 3 in [16]. The latter result is very near to the real frontier between decidability and undecidability or, in other terms, between the possibility and the impossibility of predicting the eventual behaviour of such systems, depending on the number of test tubes. Indeed, as it is known that for a single test tube, generated languages are regular [15], it remains to examine the case of two test tubes, which is still open up to now.

Time-varying distributed H systems were recently introduced in [12], [13], [14] as another theoretical model of biomolecular computing (DNA-computing), based on splicing operations. Instead of considering *test tubes*, these models introduce *components*, later see the formal definition, which cannot all be used at the same time but one after another, periodically.

This aims at giving an account of real biochemical reactions where the work of enzymes essentially depends on the environment conditions. In particular, at any moment, only a subset of all available rules are in action. If the environment is periodically changed, then the active enzymes change also periodically.

In [13], it is proved that 7 different *components* are enough in order to generate any recursively enumerable language. Recently, see [6,7], both authors proved that two components are enough to construct a *universal* time-varying distributed H system, *i.e.* a time-varying distributed H system, capable of simulating the computation of any Turing machine. Universality of computation and generating any recursively enumerable language are equivalent properties, but it is *a priori* not necessarily true, that the universality of some time-varying distributed H system with two components entails that time-varying distributed H systems generate all recursively enumerable languages, with only two components. We proved in [8] that this is the case: 2 different components are enough in order to generate any recursively enumerable language. That result was presented by both authors at FBDU'98, satellite workshop of MFCS/CSL'98 Federal Conference, August 22 - 30, 1998, Brno, Czech Republic. The proof was made by modelling Turing machines and, within that frame, every language is generated "step by step" or "word by word".

In the meanwhile in [9], it was proved that time-varying distributed H systems with 4 components generate all recursively enumerable languages. That solution by A.Paun is considered as a "parallel" one. Indeed, although a formal definition of a parallel DNA computation is not known, it is generally agreed that if a given family of systems is able to mimic any *type 0* formal grammar, such a computation can be considered as a paradigm of parallel computation. So, A.Paun's result about TVDH systems of degree 4 can be considered as a universal parallel DNA computation.

Now we improved A.Paun's result by reducing the number of components of TVDH systems which model *type 0* formal grammars down to 3. The corresponding question for 2 components remains open.

We consider also G.Paun's original definition of TVDH systems and suggest a slightly different definition of such systems based on the definition of H systems (so-called extended TVDH systems or ETVDH systems). For ETVDH systems

we proved that these systems with one component generate exactly the set of all regular languages and with two components exactly the set of all recursively enumerable languages.

Accordingly, our paper settles a frontier between decidability and undecidability for extended time-varying distributed H systems in terms of the number of their components: two is the smallest required number of components in order to generate all recursively enumerable sets.

The question of the status of time-varying distributed H system of degree one is still open.

We consider the "parallel" and "nonparallel" aspects also for ETVDH systems. Our solution for ETVDH systems of degree 2 is the same as for TVDH systems of degree 2 and so it is not a "parallel" one. Therefore there is a question: how many components are needed for ETVDH systems to generate any recursively enumerable languages in a parallel fashion? A partial answer is given in [19]: ETVDH systems of degree 4 generate any recursively enumerable language by modelling *type 0* formal grammar. Accordingly, that solution is "parallel".

2 Basic Definitions

We recall some notions. An *alphabet* V is a finite, non-empty set whose elements are called *letters*. A *word* (over some alphabet V) is a finite (possibly empty) concatenation of letters (from V). The empty concatenations of letters is also called the *empty word* and is denoted by ε . The set of all words over V is denoted by V^* . A *language* (over V) is a set of words (over V).

A *formal grammar* (of *type 0*) G is a tuple $G = (N, T, P, S)$ of an alphabet N of so-called *non-terminal* letters, an alphabet T of so-called *terminal* letters, with $N \cap T = \emptyset$, an *initial letter* S from N , and a finite set P of *rules* of the form $u \rightarrow v$ with $u, v \in (N \cup T)^*$. Note that u contains at least one symbol from N . Any rule $u \rightarrow v \in P$ is a substitution rule allowing to substitute any occurrence of u in some word w by v .

Formally, we write $w \Rightarrow_G w'$ if there is a rule $u \rightarrow v$ in P and words $w_1, w_2 \in (N \cup T)^*$ with $w = w_1 u w_2$ and $w' = w_1 v w_2$. We denote by \Rightarrow_G^* the reflexive and transitive closure of \Rightarrow . I.e., $w \Rightarrow_G^* w'$ means that there is an integer n and words w_1, \dots, w_n with $w = w_1, w' = w_n$ and $w_i \Rightarrow_G w_{i+1}$ for all $i, 1 \leq i < n$.

The sequence $w_1 \Rightarrow w_2 \Rightarrow \dots \Rightarrow w_n$ is also called a *computation* (from w_1 to w_n of length $n - 1$). A *terminal word* is a word in T^* ; all terminal words computable from the initial letter S form the language $L(G)$ generated by G . More formally, $L(G) \stackrel{\text{def}}{=} \{w \in T^*; S \Rightarrow^* w\}$.

An (*abstract*) *molecule* is simply a word over some alphabet. A *splicing rule* (over alphabet V), is a quadruple (u_1, u_2, u'_1, u'_2) of words $u_1, u_2, u'_1, u'_2 \in V^*$, which is often written in a two dimensional way as follows: $\frac{u_1 | u_2}{u'_1 | u'_2}$.

A splicing rule $r = (u_1, u_2, u'_1, u'_2)$ is applicable to two molecules m_1, m_2 if there are words $w_1, w_2, w'_1, w'_2 \in V^*$ with $m_1 = w_1 u_1 u_2 w_2$ and $m_2 = w'_1 u'_1 u'_2 w'_2$,

and produces two new molecules $m'_1 = w_1 u_1 u'_2 w'_2$ and $m'_2 = w'_1 u'_1 u_2 w_2$. In this case, we also write $(m_1, m_2) \vdash_r (m'_1, m'_2)$.

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

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

$$\sigma_h(L) = \sigma_{(V,R)}(L) \stackrel{\text{def}}{=} \{w, w' \in V^* \mid \exists w_1, w_2 \in L : \exists r \in R : (w_1, w_2) \vdash_r (w, w')\}.$$

A *Head-splicing-system* [34], or *H system*, is a construct:

$$H = (h, A) = ((V, R), A),$$

which consists of an alphabet V , a set $A \subseteq V^*$ of initial molecules over V , the *axioms*, and a set $R \subseteq V^* \times V^* \times V^* \times V^*$ of splicing rules. H is called finite if A and R are finite sets.

For any H scheme h and any language $L \in V^*$ we define:

$$\begin{aligned} \sigma_h^0(L) &= L, \\ \sigma_h^{i+1}(L) &= \sigma_h^i(L) \cup \sigma_h(\sigma_h^i(L)), \\ \sigma_h^*(L) &= \bigcup_{i \geq 0} \sigma_h^i(L). \end{aligned}$$

The language generated by H system H is defined as $L(H) \stackrel{\text{def}}{=} \sigma_h^*(A)$.

Thus, the language generated by H system H is the set of all molecules that can be generated in H starting with A as initial molecules by iteratively applying splicing rules to copies of the molecules already generated.

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

$$D = (V, T, A, R_1, R_2, \dots, R_n),$$

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

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

$$\begin{aligned} L_1 &= A, \\ L_{k+1} &= \sigma_{h_i}(L_k), \text{ for } i \equiv k(\text{mod } n), \ k \geq 1, 1 \leq i \leq n, \ h_i = (V, R_i). \end{aligned}$$

Therefore, from a step k to the next step, $k + 1$, one passes only the result of splicing the strings in L_k according to the rules in R_i for $i \equiv k(\text{mod } n)$; the strings in L_k that cannot enter a splicing are removed.

The language generated by D is, by definition:

$$L(D) \stackrel{\text{def}}{=} (\bigcup_{k \geq 1} L_k) \cap T^*.$$

We denote by *REG* the set of all regular languages, by *RE* the set of all recursively enumerable languages, by *VDH_n*, $n \geq 1$, the family of languages generated by time-varying distributed H systems of degree at most n , and by *VDH_{*}* the family of all languages of this type.

3 TVDH Systems of Degree 3

Theorem 1. *For any type 0 formal grammar $G = (N, T, P, S)$ there is a TVDH system $D_G = (V, T, A, R_1, R_2, R_3)$ of degree 3 which simulates G and $L(G) = L(D_G)$.*

Proof. We construct $D_G = (V, T, A, R_1, R_2, R_3)$ as follows. Let be $N \cup T \cup \{B\} = \{a_1, a_2, \dots, a_n\}$ ($a_n = B$) and $B \notin \{N \cup T\}$.

Alphabet $V = N \cup T \cup \{B\} \cup \{X, Y, Z, X_i, Y_i, X'_j, Y'_j, X''_j, Y''_j, X', Y', Z', Z''\}$, $1 \leq i \leq n$, $1 \leq j \leq n-1$.

The terminal alphabet is T , the same as for the formal grammar G .

Axioms $A = \{XSBY\} \cup \{ZvY_i, \exists u : u \rightarrow va_i \in P, i \in \{1, \dots, n-1\}\} \cup \{ZY_i, X'_jZ, X'Z, X_ia_iZ, ZY''_j, X_jZ, XZ, ZY'_j, ZY', X''_jZ, ZY, Z', Z'', 1 \leq i \leq n, 1 \leq j \leq n-1\}$.

Component R_1 .

$$1.1 : \frac{\varepsilon | uY}{Z | vY_i} \quad \exists u : u \rightarrow va_i \in P, i \in \{1, \dots, n-1\},$$

$$1.2 : \frac{\varepsilon | a_iY}{Z | Y_i} \quad 1 \leq i \leq n,$$

$$1.3 : \frac{X_j | \varepsilon}{X'_{j-1} | Z} \quad 2 \leq j \leq n, \quad 1.4 : \frac{X_1 | \varepsilon}{X' | Z}$$

$$1.5 : \frac{\varepsilon | Y''_j}{Z | Y_j} \quad 1 \leq j \leq n-1,$$

$$1.6 : \frac{\varepsilon | a_iuY}{Z | Y_i} \quad \exists u : u \rightarrow \varepsilon \in P, 1 \leq i \leq n,$$

Component R_2 .

$$2.1 : \frac{X | \varepsilon}{X_ia_i | Z} \quad 1 \leq i \leq n, \quad 2.2 : \frac{\varepsilon | Y'_j}{Z | Y''_j} \quad 1 \leq j \leq n-1,$$

$$2.3 : \frac{X''_j | \varepsilon}{X_j | Z} \quad 1 \leq j \leq n-1, \quad 2.4 : \frac{X' | \varepsilon}{X | Z} \quad 2.5 : \frac{X' | \varepsilon}{\varepsilon | Z'}$$

Component R_3 .

$$3.1 : \frac{\varepsilon | Y_j}{Z | Y'_{j-1}} \quad 2 \leq j \leq n, \quad 3.2 : \frac{X'_j | \varepsilon}{X_j | Z} \quad 1 \leq j \leq n-1,$$

$$3.3 : \frac{\varepsilon | Y_1}{Z | Y'} \quad 3.4 : \frac{\varepsilon | Y'}{Z | Y} \quad 3.5 : \frac{\varepsilon | BY'}{Z'' | \varepsilon}$$

Note.

Components R_1, R_2 and R_3 also contain the following rules:

$$\frac{\alpha | \varepsilon}{\alpha | \varepsilon} \quad \text{for each axiom } \alpha \in A, \text{ except } XSBY.$$

Details.

Our simulation is based on the method of rotating words [10] and the corresponding technique that are proposed by G.Paun [13]. Let us recall them briefly.

For any word $w = w'w'' \in (N \cup T)^*$ of formal grammar G say that word $Xw''Bw'Y$ ($X, Y, B \notin N \cup T$) of TVDH system D_G is a "rotation version" of word w . Say also that going from a rotation version of the word to another one is rotating the word. TVDH system D_G models the formal grammar G as follows. System D_G rotates the word Xw_1uw_2BY into Xw_2Bw_1uY . And so rotating is used to put the occurrence of u to the end of the word. After that D_G applies a splicing rule $\frac{\varepsilon|uY}{Z|vY_i}$. So we model the application of a rule $u \rightarrow va_i$ of a formal grammar G by a single scheme-rule. TVDH system D_G rotates the word Xwa_iY ($a_i \in (N \cup T \cup \{B\})$) "symbol by symbol", i.e. word Xa_iwY is obtained after performing a few steps of system D_G . Rotating a word can be done as follows. We start with word Xwa_iY in component R_1 . Component R_2 receives the word XwY_i from component R_1 . Component R_3 receives words $X_ja_jwY_i$ ($1 \leq j \leq n$) from component R_2 . After that point system D_G works in a cycle in which indexes i and j decrease. If $j \neq i$, then derived words with these indexes will be ruled out. When $i = j$ we obtain word $X_1a_iwY_1$. Then, from that word we obtain word Xa_iwY (so we rotated the word Xwa_iY) and word a_iw' (if $a_iw = a_iw'B$). So, if $a_iw' \in T^*$ then $a_iw' \in L(G)$ and $a_iw' \in L(D_G)$.

(i) $L(G) \subseteq L(D_G)$.

We start with word $XwY = Xw'a_iY$ ($w \in (N \cup T \cup \{B\})^*$, $1 \leq i \leq n$).

$(Xw|uY, Z|vY_i) \vdash_{1.1} (XwvY_i, ZuY)$, $i \in \{1, \dots, n-1\}$.

String ZuY cannot enter a splicing in R_2 and therefore is removed. So we model the application of rule $u \rightarrow va_i$.

$(Xw'|a_iuY, Z|Y_i) \vdash_{1.6} (Xw'Y_i, Za_iuY)$, $1 \leq i \leq n$.

String Za_iuY cannot enter a splicing in R_2 and therefore is removed. So we model the application of rule $u \rightarrow \varepsilon$.

$(Xw'|a_iY, Z|Y_i) \vdash_{1.2} (Xw'Y_i, Za_iY)$, $1 \leq i \leq n$.

String Za_iY cannot enter a splicing in R_2 and therefore is removed. So we start to rotate the word $w'a_i$.

$(X|w'Y_i, X_ja_j|Z) \vdash_{2.1} (\mathbf{X_j a_j w' Y_i}, XZ)$, $1 \leq i, j \leq n$.

String XZ is an axiom.

$(X_ja_jw'|Y_i, Z|Y'_{i-1}) \vdash_{3.1} (X_ja_jw'Y'_{i-1}, ZY_i)$, $1 \leq j \leq n$, $2 \leq i \leq n$.

String ZY_i is an axiom.

$(X_j|a_jw'Y'_{i-1}, X'_{j-1}|Z) \vdash_{1.3} (X'_{j-1}a_jw'Y'_{i-1}, X_jZ)$, $2 \leq i, j \leq n$.

String X_jZ is an axiom.

$(X'_{j-1}a_jw'|Y'_{i-1}, Z|Y''_{i-1}) \vdash_{2.2} (X'_{j-1}a_jw'Y''_{i-1}, ZY'_{i-1})$, $2 \leq i, j \leq n$.

String ZY'_{i-1} is an axiom.

$(X'_{j-1}|a_jw'Y''_{i-1}, X''_{j-1}|Z) \vdash_{3.2} (X''_{j-1}a_jw'Y''_{i-1}, X'_{j-1}Z)$, $2 \leq i, j \leq n$.

String $X'_{j-1}Z$ is an axiom.

$(X''_{j-1}a_jw'|Y''_{i-1}, Z|Y_{i-1}) \vdash_{1.5} (X''_{j-1}a_jw'Y_{i-1}, ZY''_{i-1})$, $2 \leq i, j \leq n$.

String ZY''_{i-1} is an axiom.

$(X''_{j-1}|a_jw'Y_{i-1}, X_{j-1}|Z) \vdash_{2.3} (\mathbf{X_{j-1} a_j w' Y_{i-1}}, X''_{j-1}Z)$, $2 \leq i, j \leq n$.

String $X''_{j-1}Z$ is an axiom.

And so on.

There are 3 cases:

- 1) $\mathbf{X_1 a_j w' Y_k}$ 2) $\mathbf{X_k a_j w' Y_1}$, $2 \leq k \leq n$, and 3) $\mathbf{X_1 a_i w' Y_1}$.

Case 1):

$$(X_1 a_j w' | Y_k, Z | Y'_{k-1}) \vdash_{3.1} (X_1 a_j w' Y'_{k-1}, Z Y_k), 2 \leq k \leq n.$$

String $Z Y_k$ is an axiom.

$$(X_1 | a_j w' Y'_{k-1}, X' | Z) \vdash_{1.4} (X' a_j w' Y'_{k-1}, X_1 Z), 2 \leq k \leq n.$$

String $X_1 Z$ is an axiom.

$$(X' a_j w' | Y'_{k-1}, Z | Y''_{k-1}) \vdash_{2.2} (X' a_j w' Y''_{k-1}, Z Y'_{k-1}), 2 \leq k \leq n.$$

String $Z Y'_{k-1}$ is an axiom. String $X' a_j w' Y''_{k-1}$ cannot enter a splicing in R_3 and therefore is removed.

$$(X' | a_j w' Y'_{k-1}, X | Z) \vdash_{2.4} (X a_j w' Y'_{k-1}, X' Z), 2 \leq k \leq n.$$

String $X' Z$ is an axiom. String $X a_j w' Y'_{k-1}$ cannot enter a splicing in R_3 and therefore is removed.

$$(X' | a_j w' Y'_{k-1}, | Z') \vdash_{2.5} (a_j w' Y'_{k-1}, X' Z'), 2 \leq k \leq n.$$

Strings $a_j w' Y'_{k-1}$ and $X' Z'$ cannot enter a splicing in R_3 and therefore are removed.

So, this computation is "mortal", i.e. it completes without results.

Case 2):

$$(X_k a_j w' | Y_1, Z | Y') \vdash_{3.3} (X_k a_j w' Y', Z Y_1), 2 \leq k \leq n.$$

String $Z Y_1$ is an axiom.

$$(X_k | a_j w' Y', X'_{k-1} | Z) \vdash_{1.3} (X'_{k-1} a_j w' Y', X_k Z), 2 \leq k \leq n.$$

String $X_k Z$ is an axiom.

String $X'_{k-1} a_j w' Y'$ cannot enter a splicing in R_2 and therefore is removed.

So, this computation also is "mortal".

Case 3):

$$(X_1 a_i w' | Y_1, Z | Y') \vdash_{3.3} (X_1 a_i w' Y', Z Y_1), 1 \leq i \leq n.$$

String $Z Y_1$ is an axiom.

$$(X_1 | a_i w' Y', X' | Z) \vdash_{1.4} (X' a_i w' Y', X_1 Z), 1 \leq i \leq n.$$

String $X_1 Z$ is an axiom.

$$(X' | a_i w' Y', X | Z) \vdash_{2.4} (X a_i w' Y', X' Z), 1 \leq i \leq n.$$

String $X' Z$ is an axiom.

$$(X' | a_i w' Y', | Z') \vdash_{2.5} (a_i w' Y', X' Z'), 1 \leq i \leq n.$$

String $X' Z'$ cannot enter a splicing in R_3 and therefore is removed.

$$(X a_i w' | Y', Z | Y) \vdash_{3.4} (X a_i w' Y, Z Y'), 1 \leq i \leq n.$$

String $Z Y'$ is an axiom.

And now, $X a_i w' Y$ is the expected result. We rotated symbol a_i from the right side of the developed word to the left side one.

$$(Xa_iw''|BY', Z'') \vdash_{3.5} (Xa_iw'', Z''BY'), 1 \leq i \leq n.$$

Strings Xa_iw'' and $Z''BY'$ cannot enter a splicing in R_1 and therefore are removed.

$$(a_iw''|BY', Z'') \vdash_{3.5} (a_iw'', Z''BY'), 1 \leq i \leq n.$$

String $Z''BY'$ cannot enter a splicing in R_1 and therefore is removed.

And now, a_iw'' may be the expected result (a pure terminal string). If $a_iw'' \in T^*$ and so $a_iw'' \in L(G)$, then $a_iw'' \in L(D_G)$ also and this part of theorem 1 is proved.

D_G continues its computations.

$$(a_iw'|Y', Z|Y) \vdash_{3.4} (a_iw'Y, ZY'), 1 \leq i \leq n.$$

String ZY' is an axiom.

Rule 1.1 may be applied:

$$(a_iw''|uY, Z|vY_k) \vdash_{1.1} (a_iw''vY_k, ZuY), 1 \leq i \leq n, k \in \{1, \dots, n-1\}.$$

Strings $a_iw''vY_k$ and ZuY cannot enter a splicing in R_2 and therefore are removed.

Rule 1.6 may be applied:

$$(a_iw''|a_luY, Z|Y_l) \vdash_{1.6} (a_iw''Y_l, Za_luY), 1 \leq i, l \leq n.$$

Strings $a_iw''Y_l$ and Za_luY cannot enter a splicing in R_2 and therefore are removed.

$$(a_iw''|a_lY, Z|Y_l) \vdash_{1.2} (a_iw''Y_l, Za_lY), 1 \leq i, l \leq n.$$

Strings $a_iw''Y_l$ and Za_lY cannot enter a splicing in R_2 and therefore are removed.

So, these computations are "mortal" too and D_G cannot produce any other molecules than a_iw'' .

(ii) $L(D_G) \subseteq L(G)$.

Let us examine the behaviour of D_G . First of all, we can see that D_G correctly simulates the use of any rule $u \rightarrow v \in P$, because all "superfluous" molecules are ruled out. We note that the subscripts of X, Y can be removed only when $i = j$ during the simulation of rule $u \rightarrow v \in P$. More generally, occurring non-terminal letters can be removed to obtain the "pure" terminal word only when we simulates the rule $u \rightarrow v \in P$. So, if $w \in L(D_G)$ then $w \in L(G)$. \square

4 Modified Definition of TVDH Systems – Extended TVDH Systems

We extend the definition of H system to components of TVDH system, so the components $R_i, 1 \leq i \leq n$ of TVDH system of degree n will work as corresponding H systems.

Let us define operation $\tilde{\sigma}_h(L)$ as follows:

$$\tilde{\sigma}_h(L) = \tilde{\sigma}_h(L' \cup L'') \stackrel{\text{def}}{=} \sigma_h^*(L'), \text{ where}$$

$$L' = \{w_1 \in L | \exists w_2 \in L : \exists w, w' \in V^* : \exists r \in R : (w_1, w_2) \vdash_r (w, w') \text{ or } (w_2, w_1) \vdash_r (w, w')\}, L'' = L \setminus L', \text{ and } h = (V, R).$$

(We note that every language $L \subseteq V^*$ for every H scheme h can be split recursively into two subsets L' and L'').

So, $\tilde{\sigma}_h(L)$ works as H system $(h, L') = ((V, R), L')$.

An *extended time-varying distributed H system* (of degree n , $n \geq 1$), (ETVDH system) is a construct:

$$E = (V, T, A, R_1, R_2, \dots, R_n),$$

where V is an alphabet, $T \subseteq V$ is the *terminal alphabet*, $A \subseteq V^*$ is the finite set of *axioms*, and R_i are *components*, i.e. finite sets of splicing rules over V , $1 \leq i \leq n$.

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

$$L_1 = A,$$

$$L_{k+1} = \tilde{\sigma}_{h_i}(L_k), \text{ for } i \equiv k \pmod{n}, k \geq 1, 1 \leq i \leq n, h_i = (V, R_i).$$

The language generated by E is:

$$L(E) = (\cup_{k \geq 1} L_k) \cap T^*.$$

Let us denote by $EVDH_n$, $n \geq 1$, the family of languages generated by extended time-varying distributed H systems of degree at most n , and $EVDH_*$ the family of all languages of this type.

Theorem 2. (see [8].) *a) $EVDH_1 = REG$, b) $EVDH_2 = EVDH_* = RE$.*

Proof. Let be $E = (V, T, A, R_1)$. As there is a single component, we can denote it simply R and, accordingly, $\tilde{\sigma}_{h_1}$ can also be denoted by $\tilde{\sigma}$ and $\sigma_{h_1}^*$ by σ^* . According to the definitions, $L_1 = A$. We have that $L_2 = \tilde{\sigma}(A)$.

Let us now split L_2 into two subsets: L'_2 will denote the words which enter the rules of the component and L''_2 will denote those which do not enter the rules. Accordingly, $L_3 = \tilde{\sigma}(L_2) = \sigma^*(L'_2)$. By construction, we have that $L'_2 \subseteq L_2$. Split L_3 into two subsets L'_3 and L''_3 defined in the same way as for L_2 . We have that $L_3 = \tilde{\sigma}(L_2) = \sigma^*(L'_2)$ and, as $L'_2 \subseteq L_2$, that $L'_3 \subseteq L'_2$.

Now, $L_4 = \tilde{\sigma}(L_3) = \sigma^*(L'_3) \subseteq \sigma^*(\sigma^*(L'_2))$. But, by the definition of σ^* , it is plain that $\sigma^*(\sigma^*(L)) = \sigma^*(L)$ for any language L . Accordingly, $L_4 \subseteq L_3$. And so, due to the definition of $L(E)$, $L(E) = (L_1 \cup L_2 \cup L_3) \cap T^*$.

By [15], we know that extended H systems with a finite set of axioms and a finite set of rules generate regular languages. Accordingly, L_2 is a regular language. For each rule r of R , let us define L_r as the set of words upon V^* which do not enter r . As it is plain that the complement of L_r in V^* is recognizable by a finite automaton, L_r is a regular language and so, as R is finite and as $L'_2 = \bigcup_{r \in R} (L_2 \cap L_r)$, L'_2 is also a regular language. As L'_2 is the complement of L''_2 in L_2 which is also a regular language, we obtain that L'_2 is a regular language. By [11], extended H systems with a regular set of axioms and with a finite number of rules generate regular languages. Accordingly, L_3 is a regular language, and so is $L(E)$. This proves that $L(E) \subseteq REG$. Similar as [13], it is easy to prove that $REG \subseteq EVDH_1$, so the proof of theorem 2 a) is completed.

The part b) is true immediately after a small modification of the construction of theorem 1b [8]. \square

Acknowledgement. The authors acknowledge the very helpful contribution of *INTAS project 97-1259* for enhancing their cooperation, giving the best conditions for producing the present result. For the same reasons they also acknowledge the help of the University of Metz (France).

References

1. Csuhaaj-Varjù, E., Kari, L., Păun, G.: Test Tube distributed system based on splicing. *Computer and AI*. 2–3 (1996) 211–232
2. Ferretti, C., Mauri, G., Zandron, C.: Nine test tubes generate any RE language. *TCS*. **231**, no.2 (2000) 171–180
3. Head, T.: Formal language theory and DNA: an analysis of the generative capacity of recombinant behaviors. *Bulletin of Mathematical Biology* **49** (1987) 737–759
4. Head, T., Păun, Gh., Pixton, D.: Language theory and molecular genetics. Generative mechanisms suggested by DNA recombination. Chapter 7 in vol.2 of G.Rozenberg, A.Salomaa, eds., *Handbook of Formal Languages*, 3 volumes, Springer-Verlag, Heidelberg (1997)
5. Margenstern, M.: Frontier between decidability and undecidability: a survey. *TCS*. **231**, no.2 (2000) 217–251
6. Margenstern, M., Rogozhin, Yu.: A universal time-varying distributed H -system of degree 2. In Preliminary proceedings, Fourth International Meeting on DNA Based Computers, June 15-19, 1998, University of Pennsylvania (1998) 83–84
7. Margenstern, M., Rogozhin, Yu.: A universal time-varying distributed H -system of degree 2. *Biosystems* **52** (1999) 73–80
8. Margenstern, M., Rogozhin, Yu.: Generating all recursively enumerable languages with a time-varying distributed H -system of degree 2. Publications du GIFM, no. 99-102, I.U.T., Metz (France), ISBN 2-9511539-5-3 (1999)
9. Păun, A.: On Time-Varying H Systems. *Bulletin of EATCS*. **67** (1999) 157–164
10. Păun, G.: Regular extended H systems are computationally universal. *Journal of Automata, Languages and Combinatorics* **1**, no.1 (1996) 27–36
11. Păun, G., Rozenberg, G., Salomaa, A.: Computing by splicing. *TCS*. **168**, no.2 (1996) 321–336
12. Păun, G.: DNA computing: distributed splicing systems. In *Structures in Logic and Computer Science. A Selection of Essays in honor of A. Ehrenfeucht*, LNCS. **1261** (1997) 353–370
13. Păun, G.: DNA Computing Based on Splicing: Universality Results. *TCS*. **231**, no.2 (2000) 275–296
14. Păun, G., Rozenberg, G., Salomaa, A.: *DNA Computing: New Computing Paradigms*. Springer, Heidelberg (1998)
15. Pixton, D.: Regularity of splicing languages. *Discrete Applied Mathematics* **69** (1996) 101–124
16. Priesse, L., Rogozhin, Yu., Margenstern, M.: Finite H -Systems with 3 Test Tubes are not Predictable. In *Proceedings of Pacific Symposium on Biocomputing*, Kapalua, Maui, January 1998 (R.B.Altman, A.K.Dunker, L.Hunter, T.E.Klein, eds), World Sci. Publ., Singapore (1998) 545–556
17. Rogozhin, Yu.: Small universal Turing machines. *TCS*. **168**, no.2 (1996) 215–240
18. Rogozhin, Yu.: A Universal Turing Machine with 22 States and 2 Symbols. *Romanian Journal of Information Science and Technology* **1**, no.3 (1998) 259–265
19. Verlan, S.: On extended time-varying distributed H systems. Poster at DNA6 conference, Leiden (2000)

String Tile Models for DNA Computing by Self-Assembly

Erik Winfree¹, Tony Eng², and Grzegorz Rozenberg^{3,4}

¹ Depts. of Computer Science and CNS,
California Institute of Technology,
Pasadena CA 91125, USA,
winfree@caltech.edu,
<http://gg.caltech.edu/~winfree>,

² Laboratory for Computer Science,
Massachusetts Institute of Technology,
Cambridge, MA 02139, USA,
tleng@theory.lcs.mit.edu,

³ Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1, 2333 CA Leiden
The Netherlands

⁴ Department of Computer Science
University of Colorado at Boulder
Boulder, CO 80309, USA.
rozenber@wi.leidenuniv.nl

Abstract. This paper investigates computation by linear assemblies of complex DNA tiles, which we call string tiles. By keeping track of the strands as they weave back and forth through the assembly, we show that surprisingly sophisticated calculations can be performed using linear self-assembly. Examples range from generating an addition table to providing $O(1)$ solutions to CNF-SAT and DHPP. We classify the families of languages that can be generated by various types of DNA molecules, and establish a correspondence to the existing classes $ET0L_{ml}$ and $ET0L_{fin}$. Thus, linear self-assembly of string tiles can generate the output languages of finite-visit Turing Machines.

1 Introduction

Adleman's original work on molecular computation [Adl94] made use of self-assembly for an important step in the computation: the generation of DNA representing paths through a graph of vertices. This is a useful preprocessing step, reducing the set of all possible sequences of vertices to just a subset (the valid paths) that can be exponentially smaller. However, linear self-assembly of double-helical DNA appeared to be limited, prompting the suggestion to use the self-assembly of two-dimensional [Win96] and branched [WYS98] DNA structures for DNA-based computation. These suggestions were predicated on the

complex synthetic DNA structures invented by Seeman for DNA nanotechnology [See82, See98]; there is now over a decade of experimental work with these molecules, including the recent demonstration of two-dimensional (2D) crystals and their modification [WLWS98, LYK⁺00, MSS99, LSS99].

Self-assembly and branched DNA structures may be used in combination with other DNA computing techniques. Reif has proposed using step-wise self-assembly to reduce errors [Rei99]; the circuit satisfaction problem has a particularly elegant implementation in his model. Jonoska has considered the self-assembly of branched DNA into flexible graph-like structures, with applications to NP-complete problems [JKS99]. An interesting observation is that pre-formed branched DNA structures can provide advantages for subsequent processing by restriction enzyme digestion [JKS98]. Even knottedness can be used for computation; DNA Borromean rings implement a logical AND gate [SWY⁺98]. We are a long way from understanding the full power of branched DNA and self-assembly.

However, these proposals still use complex DNA structures and assemblies that are likely to pose at least as many technical difficulties as 2D self-assembly. It has been suggested that fixed-width or one-dimensional (1D) self-assembly may be an attractive and robust experimental system, with faster self-assembly and lower error rates than two-dimensional systems [LWR00]. In this article we explore the computational power of 1D self-assembly of branched DNA structures.

2 DNA Self-Assembly and Formal Language Theory

In DNA-based computing, a test tube of DNA oligonucleotides (equivalently, strands) is considered to represent a set of strings over an alphabet. A strand of DNA can be interpreted directly as a string over $\mathcal{D} = \{A, C, G, T\}$ (a DNA sequence) by reading its bases in the $5' \rightarrow 3'$ order. If S is a DNA sequence, then its Watson-Crick complement is written S' ; e.g. $AGCTGCG' = CGCAGCT$. We will follow the convention that DNA strands may be taken to represent strings over a larger alphabet Σ by using a codebook $\mathcal{C} : \Sigma \rightarrow \mathcal{D}^N$; i.e., each symbol in Σ is represented by an N -base subsequence. It will always be assumed that for $\alpha \neq \beta$, occurrences of $\mathcal{C}(\alpha)$ and $\mathcal{C}(\beta)$ are guaranteed not to overlap in the DNA strands under consideration. Thus, a tube of DNA strands can be considered to represent a set of strings over the alphabet Σ . Because DNA strands can be circular, a test tube may also contain circular strings over \mathcal{D} and Σ , represented using the prefix symbol \circ as described later. Finally, note that the codebook defines a many-to-one relationship of \mathcal{D}^* to Σ^* , because we use the convention that DNA subsequences that are not part of any codeword $\mathcal{C}(\alpha)$ are simply ignored. For example, using a codebook where $\mathcal{C}(a) = ACT$, $\mathcal{C}(b) = GAC$, both the strings $ACTGACGAC$ and $GTACTTTGGACGTGAC$ code for abb .

Formal languages, central to understanding computation on strings, provide a natural formalism for DNA based computing. There is a close correspondence between generative grammars and the self-assembly and ligation of DNA molecules. Both are processes that generate new strings from previous ones according to

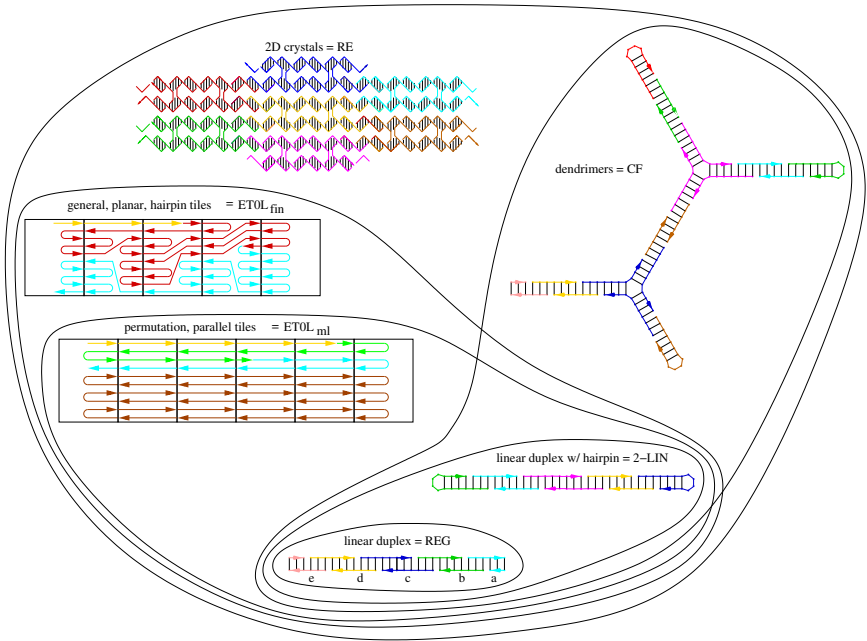


Fig. 1. The hierarchy of self-assembled languages, including the new results in this paper. Here *REG* is the family of regular languages, *LIN* is the family of linear languages (and *k-LIN* is the *k*-metalingual languages, where the axioms may have *k* non-terminal symbols), *CF* is the family of context-free languages, *CS* is the family of context-sensitive languages, and *RE* is the family of recursively enumerable languages.

well-defined rules. As an example, we informally describe the self-assembly model of Winfree et al. [WYS98]. A multi-strand DNA molecule is represented as a graph, called a *DNA complex*, where each node (labelled from \mathcal{D}) represents a nucleotide, directed *backbone* edges represent covalent phospho-diester bonds, and undirected *basepair* edges represent Watson-Crick base pairing. Self-assembly at “temperature” \mathcal{T} starts with (an unlimited supply of) a finite set of initial DNA complexes. Two DNA complexes with complementary sticky ends (of length $\geq \mathcal{T}$) can be joined to make a new complex. A DNA complex that cannot participate in further assembly is called a *maximal* (or *terminal*) complex. The set of strands remaining after ligation of all nicks in all maximal complexes is a DNA sequence language over \mathcal{D} , encoding (via the codebook \mathcal{C}) a language over Σ . For example, in the linear duplex assembly of Figure 1, the DNA sequences after ligation are the two strands $\{a b c d e, e' d' c' b' a'\}$.

Throughout this text we are interested only in the maximal, not the intermediate assemblies. That is, although we postulate an unlimited supply of the initial DNA complexes, we assume all reactions go to completion, exhausting the supply of their reactants. Our motivation is partly to avoid treatment of

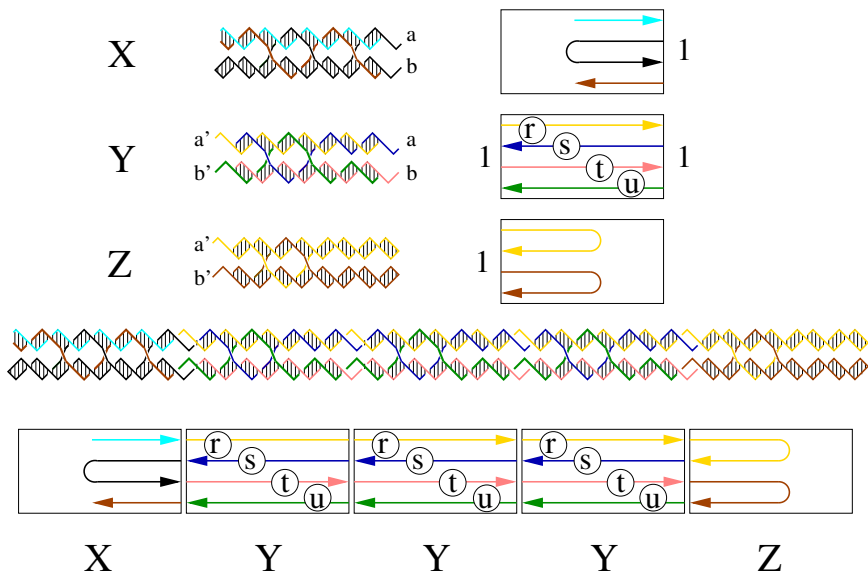


Fig. 2. Three DPE molecules that assemble to construct a language outside of context-free. Both molecular schematics and string tile diagrams are shown. Sticky-end sequences are indicated by a, b and their Watson-Crick complements a' and b' ; the matching relationship is given by the binding label 1 on the string tiles. The coding sequences in the DNA strands are indicated by the edge labels r, s, t , and u in the string tiles.

concentrations and kinetics and equilibria, as would be necessary to study intermediates, and partly because the study of maximal assemblies allows for more elegant mathematics. (More subtle models of self-assembly that attempt to treat kinetics and finite resources more realistically have proven to be mathematically challenging [Adl00].)

We are interested in computational structure-function relationships: what classes of DNA complexes (i.e. structures) give rise to what classes of languages (i.e. functions)? As is illustrated in Figure 1, the first natural classes of DNA self-assembly to be studied reproduced much of the Chomsky hierarchy for formal languages.

- Self-assembly of duplex DNA by single sticky-end adhesion generates regular languages by forming linear DNA complexes [WYS98].
- Self-assembly of hairpin and duplex DNA by single sticky-end adhesion generates 2-metalinear languages by forming linear DNA complexes (a modest generalization of [Eng99]).
- Self-assembly of hairpin, duplex, and 3-arm DNA by single sticky-end adhesion generates context-free languages by forming dendrimer DNA complexes [WYS98].

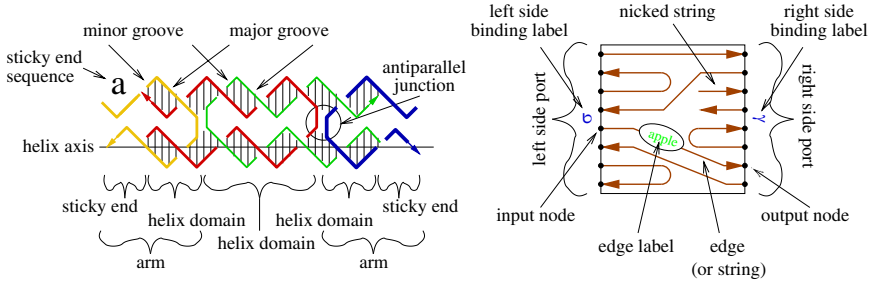


Fig. 3. Terminology for DNA multi-crossover structures and string tile diagrams. Arrowheads indicate the 3' end of the DNA. Note that in the molecular schematic (showing a DAO molecule), the major/minor groove also indicates the 3' and 5' ends: the 3' ends point away from the center of the narrow (minor) groove. The non-crossing strands in an antiparallel junction are antiparallel; in a parallel junction, they would be parallel. Hairpin strands begin and end on the same side of the tile or molecule.

- Self-assembly of DX units by double sticky-end adhesion, at a critical temperature that allows discrimination between single and double matches, can generate recursively enumerable languages by forming 2D DNA complexes [Win96, WYS98].

In this paper, we show two new classes:

- Self-assembly of DNA multi-crossover units by multiple simultaneous sticky-end adhesion generates $ETOL_{fin}$ languages by forming linear multi-helix DNA complexes.
- The above case, restricted to units where in each internal tile all coding strands cross from one side to the other side, generates $ETOL_{ml}$ languages.

The context-sensitive languages have not yet arisen in DNA self-assembly.

3 Motivating Examples

We develop the basic ideas by example, before introducing formal notation in Section 5.

3.1 A Non-context-free Language

We begin by considering linear assembly of double-crossover (DX) molecules [ES93], allowing hairpins on the arms. There are five types of DX molecules, classified according to whether the strands that don't cross at the junction are parallel (P) or anti-parallel (A) with each other, whether there are an even (E) or odd (O) number of half-turns between junctions, and whether the narrow (N) or wide (W) groove is in excess on the inside between the junctions: thus they are called DAE, DAO, DPE, DPON, DPOW.

If DPE units assemble into a linear array where each unit joins its neighbors via both of *two* sticky ends, then the resulting language of DNA sequences can be beyond context-free. Specifically, Figure 2 shows how to achieve $L = \{r^n s^n t^n u^n | n \geq 1\} \notin \mathcal{L}_{CF}$. The hairpins on the arms of X and Z allow the strands to turn around and return again through the tiles. Long-range correlations are possible due to the fact that the final DNA strand snakes through the assembly several times. The maximal DNA complexes assembled in this reaction are of the form XY^*Z , but the DNA strands encode sequences of the form $r^n s^n t^n u^n$. Thus a regular language of units yields a non-context-free language of DNA sequences.

The logic of language generation by self-assembly can be hard to see when complex multi-helical DNA structures are drawn; the situation is clarified by using (*linear*) *string tiles*, as shown in Figures 2 and 3 and described informally here (we give formal definitions in Section 5). The left and right sides are called the *ports*. Each port may be labelled by a symbol or color, called the binding label, to indicate how tiles may be joined to each other. The ports also have several input and output nodes representing the 3' and 5' ends, respectively, of strands that can be ligated to strands in adjacent tiles. The input and output nodes within a tile are connected by edges (drawn as arrows) representing the DNA strands of the tile. A missing edge (drawn as an arrow that connects to or from nothing) indicates a nick in the DNA strand. Each edge is labelled by a string over the final alphabet to indicate the coding sequence on that strand. Tiles with matching binding labels may be joined (like dominoes); assemblies which permit no further additions are called maximal assemblies. When joined, the edges in maximal assemblies form either paths or cycles; the strings labelling the edges may be concatenated to form linear or circular strings, respectively. The collection of all such concatenated linear (circular) strings, for all maximal assemblies made from a given set of tiles, is called the linear (circular) language generated by the tile set.

3.2 Parallel Tiles That Generate an Addition Table

A more interesting example, building on [Rei99, LYK+00], is generating a table of all addition input/output triples. ([LWR00] gives another implementation of this example, based on a preliminary draft of this paper.) As shown in Figure 4, the basic unit is still a DPE double crossover unit, each with a pair of sticky ends on the left and on the right. The sequences for these sticky ends are such that for any two units which bump into each other, either both sticky ends match (and the units may be joined) or both sticky ends don't match (and the units may not be joined). *Thus, a temperature that allows discrimination between a partial match and a total match is not necessary.*

How does this system work? The two possible sticky-end pairs represent the two possible carry-bit states during bitwise addition. Starting from the right, each new unit adds a new bit to \mathbf{x} and a new bit to \mathbf{y} (thus there are always 4 possibilities) and the appropriate new bit to \mathbf{z} (as a function of the previous carry and \mathbf{x} and \mathbf{y}), terminating with the sticky-end pair for the appropriate

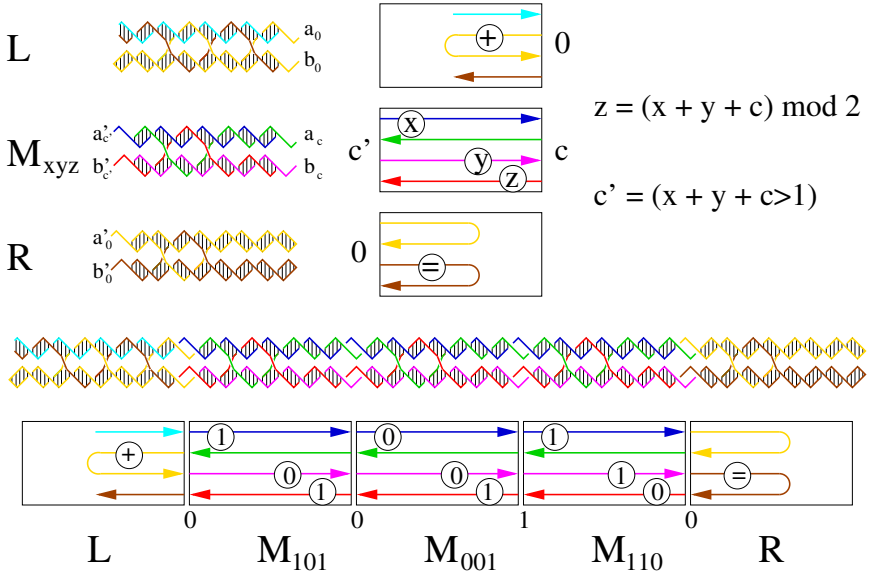


Fig. 4. Ten DPE molecules that assemble to construct a binary addition table. Here, x, y, z, c , and c' are binary variables; there is a one tile for every combination of x, y, z values. Thus, there are only four sticky-end sequences, a_0, a_1, b_0, b_1 , and their Watson-Crick complements. Likewise, there are only two binding labels, 0 and 1. Matching binding labels are indicated below the sides of joined tiles in the assembly.

carry bit (again as a function of the previous carry and \mathbf{x} and \mathbf{y}). With the capping units, the final strand through the maximal assembly zigs first through \mathbf{x} , then zags through \mathbf{y} , and finally (in reverse order) through \mathbf{z} . The generated language is thus

$$L_{ADDREV} = \{ \mathbf{x} + \mathbf{y} = \mathbf{z} : |\mathbf{x}| = |\mathbf{y}| = |\mathbf{z}| \text{ and } \# \mathbf{z}^R = \# \mathbf{x} + \# \mathbf{y} \}$$

where \mathbf{z}^R gives the string \mathbf{z} in reverse-order and $\# \mathbf{x}$ gives the integer represented by the binary string \mathbf{x} . We use quotation marks to emphasize that the contained symbols are just symbols; i.e., “+” is a symbol and not a mathematical operator in this context.

There are two potential drawbacks to the scheme illustrated in Figure 4. First, we note that [FS93] found that parallel variants (DPE, DPON, DPOW) of short double crossover molecules are less stable than the antiparallel (DAE, DAO) variants. Long arms in our DPE should stabilize the parallel structures, but it is worth investigating whether the string tiles can be implemented using only antiparallel structures. Second, the reversed output string may indicate a limitation to the string tile approach. As we see in the next example, it does not.

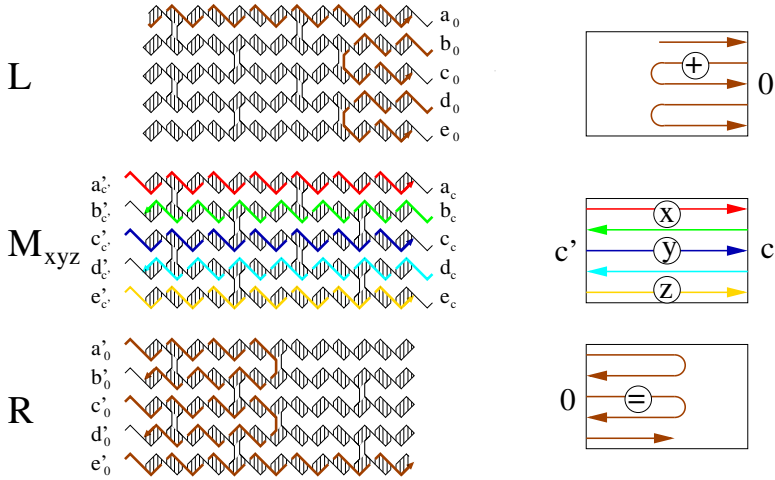


Fig. 5. Ten penta-crossover molecules that assemble to construct a binary addition table.

In fact, we can avoid both the reversed \mathbf{z} string and the parallel DX molecules by using larger multi-crossover units, thus generating exactly

$$L_{ADD} = \{“\mathbf{x} + \mathbf{y} = \mathbf{z}” : |\mathbf{x}| = |\mathbf{y}| = |\mathbf{z}| \text{ and } \#\mathbf{z} = \#\mathbf{x} + \#\mathbf{y}\}.$$

Figure 5 shows string tiles built from antiparallel quintuple-axis DNA molecules. Note that we are ignoring the black strands, which will not code for anything according to the codebook, and thus are ignored in the final language. In this system, each adhesion event now involves multiple sticky ends, but again no sensitive discrimination is required – either all sticky ends match, or none do. However, the trade-off is that more complicated DNA structures and longer stretches of non-coding DNA are required.

3.3 Hairpin Tiles for CNF-SAT

We now show a use for tiles (other than cap tiles) whose coding strands involve hairpins: they allow CNF-SAT problems to be solved in $O(1)$ biosteps using linear arrays of DNA multi-crossover units. The solution we present here, using linear self-assembly, may (or may not) be faster and more robust than the two-dimensional self-assembly of [Win96, LL00], which is also sufficient to solve this problem in $O(1)$ biosteps.

The main idea is as follows: A CNF-SAT problem of N clauses and M variables is solved using an initial set of $2M + 2$ hairpin tiles of width N , which assemble to form all 2^M distinct tile assemblies of length $M + 2$. To isolate a solution to the problem, one additional operation is required: after assembly and

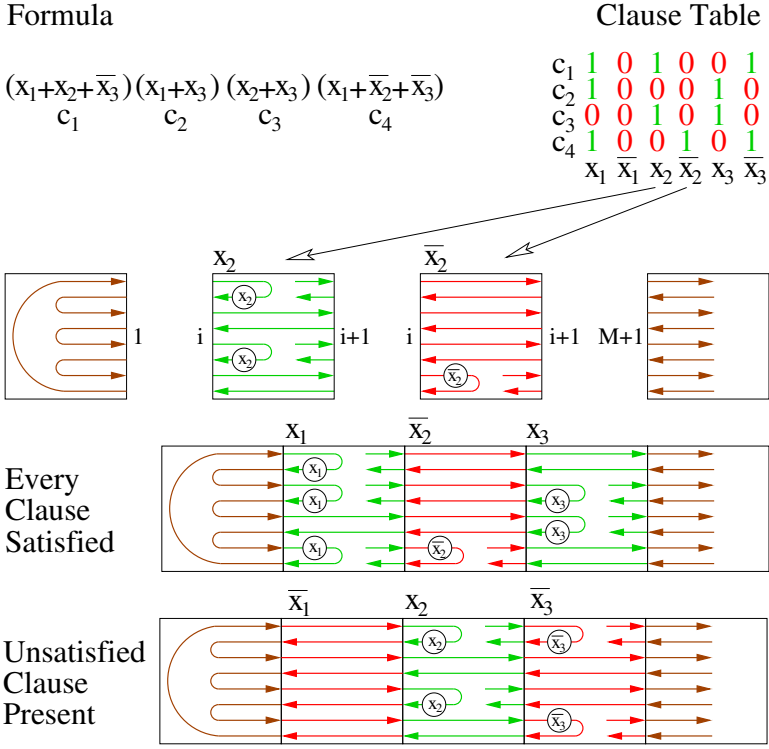


Fig. 6. Solving an N -clause M -variable CNF-SAT problem by linear assembly of $2M + 2$ width- N strings tiles. Shown are the cap tiles and the tiles for x_i and for \bar{x}_i , $i = 2$. A circular strand indicates the solution to the problem. Maximal assemblies are all of length $M + 2$.

ligation of the DNA, we select for circular strands (for example, by 2D gel electrophoresis or by exonuclease digestion). The formula is satisfiable iff a circular strand is present, as it gives a solution to the CNF-SAT problem.

More specifically, each clause C_i is a disjunction of literals chosen from $\{x_1, \dots, x_M, \bar{x}_1, \dots, \bar{x}_M\}$. The entire CNF formula, then, can be represented as a clause table C where each row signifies a clause, and each column signifies a literal. Each tile will represent a column of entries in C . Thus there are two tiles for each variable j , “True” (x_j) and “False” (\bar{x}_j). “True” has a hairpin in each row i such that the literal x_j appears in clause i , and “False” has a hairpin in each row i such that the literal \bar{x}_j appears in clause i . In any maximal assembly made from the tiles shown in Figure 6, the i^{th} row (helix axis) has a hairpin in the j^{th} column (tile) iff the assignment of “True” or “False” to variable x_j has satisfied clause i . Therefore, the strand starting in the left cap tile is circular iff every clause is satisfied. The actual satisfying assignment is deduced from the

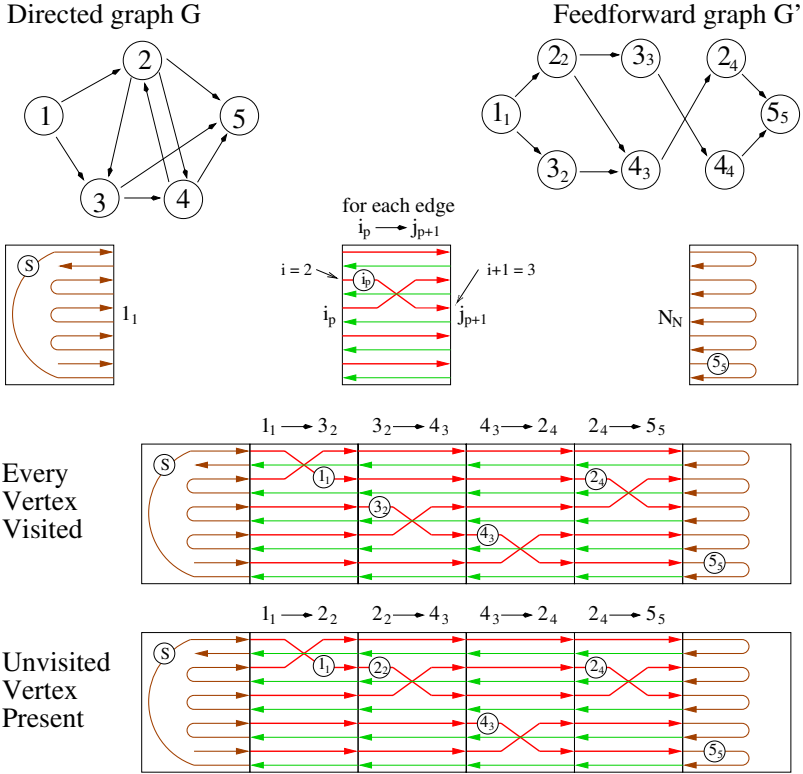


Fig. 7. $O(N^2)$ width- N strings tiles to solve an N -vertex Directed Hamiltonian Path Problem. Maximal assemblies are all of length $N + 1$.

intervening sequence. (By construction, no other strands in the assembly can be circular.)

3.4 Permutation Tiles for DHPP

As another example of solving an NP-complete problem in $O(1)$ biosteps using linear string tiles, we solve the Directed Hamiltonian Path Problem (DHPP) using permutation tiles – that is, tiles without hairpins, but where the strands may be re-routed as they cross the tile (that is, their order is permuted).

First, as a polynomial time preprocessing step, we convert the directed N -vertex graph G into a feed-forward version G' with up to N^2 vertices. The vertex at layer position p of G' corresponding to vertex i in G is labelled i_p . As shown in Figure 7, we have a permutation tile for each edge $i_p \rightarrow j_{p+1}$ in the new graph; this tile has a single pair of non-parallel arrows from left input i (respectively $i + 1$) to right output $i + 1$ (respectively i). Binding labels are such that each maximal assembly of tiles corresponds to a path from 1_1 to N_N in G' . Thus,

starting at S in a tile assembly representing a particular length- N path through G , the strand can advance from row i to row $i + 1$ only if vertex i is visited at some point. If an unvisited vertex is present, there is a row the strand cannot advance beyond, and consequently it must terminate at the 3' nick. (Note that the other strands may form a circle.) If there is no unvisited vertex, in which case the assembly represents a Hamiltonian path, the strand containing S is circular. Again, we isolate circular strands, and from those strands we further must extract strands containing S .

Note that DHPP could also be solved with hairpin tiles as was CNF-SAT, and conversely CNF-SAT could be solved with permutation tiles; this is left as an exercise to the reader.

4 Constructibility of String Tiles

The CNF-SAT and DHPP examples made use of string tiles for which no explicit DNA structures were given. Can we construct DNA molecules for the string tiles in question? We approach this first by informally defining several classes of (simpler and then more complex) string tiles, and then demonstrating a procedure to build complex string tiles from simpler ones.

4.1 Classes of String Tiles

As illustrated in Figure 8, we classify linear string tiles into *parallel tiles*, in which the i^{th} input node on one side is connected to the i^{th} output node on the other side; *permutation tiles*, in which nodes must be connected to nodes on the opposite side only, but in any order; *hairpin tiles*, which are like parallel tiles with the additional possibility that an input node may be connected to an adjacent output node on the same side; *planar tiles* in which the strands do not cross, as drawn on the tiles; and *general tiles*, in which any connections are allowed. These classes are subdivided into left and right *cap tiles*, which have one unlabelled side, and into tiles with a given number of nicks (i.e., internal 3' and 5' ends, measured in pairs).

4.2 Criteria for Constructibility; Prototiles

We might ask at this point, can general tiles of significant complexity actually be made out of DNA? To be a useful implementation of a string tile, a proposed DNA complex must

- be geometrically compatible with DNA molecular structure,
- have strand routing identical to that in the string tile,
- be “rigid” as a molecule, so that each helical domain remains parallel to the other helical domains,
- self-assemble reliably from the individual strands, when mixed according to some protocol.

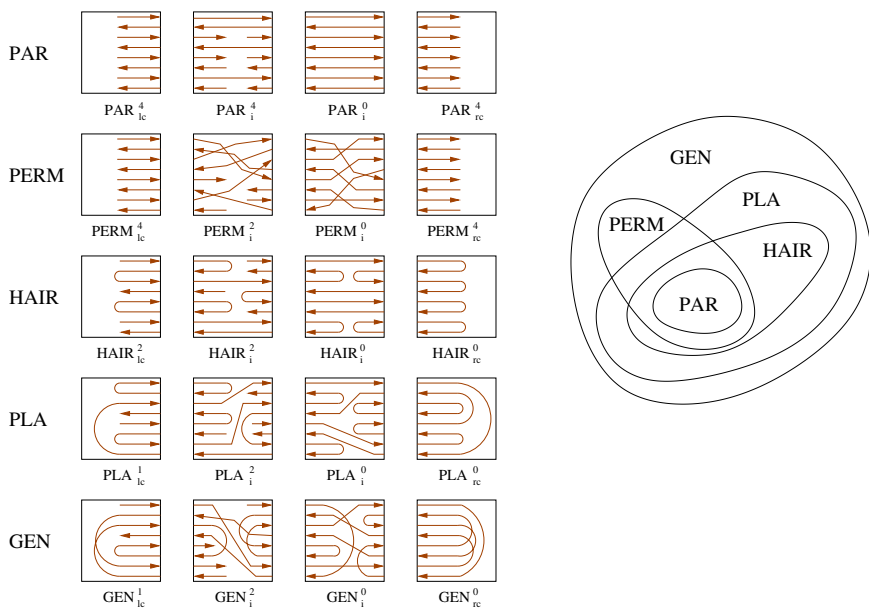


Fig. 8. The five tile classes: parallel tiles (PAR), permutation tiles (PERM), hairpin tiles (HAIR), planar tiles (PLA), and general tiles (GEN). The left (*lc*) and right (*rc*) cap tiles of each class are shown, and the number of nicks is indicated in the superscript. (left) Example tiles. (right) Tile class inclusion diagram.

For example, consider the two sets of tiles proposed for generating the addition table. In both sets, each double helix domain is connected to each neighboring domain by at least two junctions, ensuring that the axes will be parallel in the molecule and that the molecule will be rigid. The first set (Figure 4) was implemented with DPE molecules, which have been characterized in the laboratory [FS93], so they are known to self-assemble from the four component strands (although with questionable reliability). The second set (Figure 5) was implemented with hypothetical quintuple-axis molecules, which have not been experimentally demonstrated yet, although they are likely to be feasible (N. Seeman, private communication). In fact, it is unlikely that the internal tiles would spontaneously self-assemble from their 10 component strands, unless the long black strand were broken into several shorter strands; but this is a question to be answered by experiment. Direct assembly of tiles from component strands poses an even greater difficulty for larger string tiles.

Therefore, we pursue an approach where larger string tiles are assembled from a small set of *prototiles*, which consist of (or are very similar to) molecules that have already been characterized experimentally. In fact, we give two possible implementations for each prototile, one using parallel junctions (essentially DPE molecules [FS93]) and one using anti-parallel junctions (essentially triple-

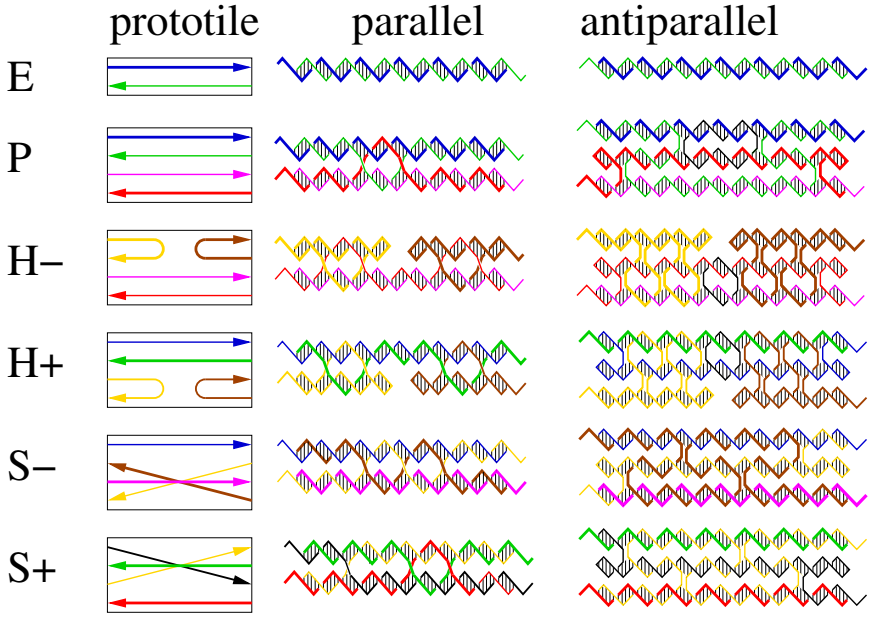


Fig. 9. Prototile sets (using only parallel junctions or only antiparallel junctions) from which all general linear string tiles can be built.

crossover molecules [\[LYK⁺00,LWR00\]](#)). Thus, the constructions below show that either parallel or anti-parallel junctions alone are sufficient for implementing all general tiles. The six prototiles are shown in Figure 9; note that they are all rigid.

4.3 Constructions

We construct a general tile in three steps. First (Figure 10a), we observe that every general tile is the composition of a permutation tile, a hairpin tile, and a permutation tile.

Second (Figure 10b), we show that any permutation tile can be built from the $E, P, S+, S-$ prototiles. To see how to arrange the prototiles, we make a list of the desired destination for the left input nodes and sort this list using the Even-Odd Transposition Sort, which is guaranteed to finish within N rounds [\[Knu73\]](#). Wherever we performed a swap, we place an $S+$ prototile; where we didn't swap, we place a P prototile; and we use the E prototile for untested positions. This correctly routes the rightward arrows, without affecting the leftward arrows. A similar procedure can be done to route the leftward arrows without altering the rightward arrows. Once the proper arrangement of the prototiles is determined, the actual DNA molecules can be made with unique sticky ends for the prototile in each position; each of these prototiles can be assembled from their component

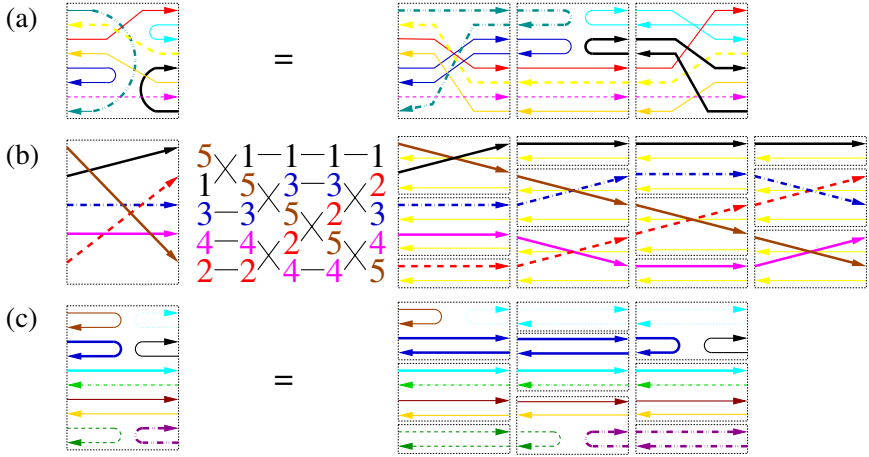


Fig. 10. (a) A general string tile with k hairpins can be created from two permutation tiles and a hairpin tile with k hairpins. (b) A permutation tile can be built from the $E, P, S+, S-$ prototiles. Here we show the construction for the rightward arrows; a similar construction routes the leftward arrows. (c) A hairpin tile with k hairpins can be construction from the $E, P, H+, H-$ prototiles.

strands in separate reactions, and then mixed and ligated to form the entire string tile.

Third (Figure 10c), we note that hairpins tiles of the form required (hairpins within a helix axis, but not between them) can be built from the $E, P, H+, H-$ prototiles. Three columns of prototiles are always sufficient.

5 Generative Power of String Tiles

We use a formal language theory approach to analyze the generative power of string tile assembly. We first give a general definition of string tiles, and then specialize to the fixed-width linear string tiles used in this paper.

5.1 Preliminaries

λ represents the empty string. From a given alphabet Φ we construct a distinct *barred* alphabet $\bar{\Phi} = \{\bar{\sigma} : \sigma \in \Phi\}$. Thus, $\Phi \cap \bar{\Phi} = \emptyset$ and, treating $\bar{\cdot}$ as an operator that is its own inverse, $\bar{\bar{\sigma}} = \sigma$. We will use Φ to represent a set of unique sticky ends, while $\bar{\Phi}$ will represent their complements.

A *circular string* over Σ is a finite set $c \subset \Sigma^+$ such that $(ab \in c \Rightarrow ba \in c)$ and $(x, y \in c \Rightarrow \exists a, b : x = ab \text{ and } y = ba)$. I.e. c consists of all circular permutations of a given string. If $x \in c$, the shorthand $\circ x$ denotes c .

A *directed graph* with edges labeled over Σ is a pair $g = (V, E)$ where $E \subseteq V \times \Sigma^* \times V$. We use the notation V_g and E_g to refer to the vertices V and edges E of g , respectively. Two graphs g_1 and g_2 are *disjoint* if $V_{g_1} \cap V_{g_2} = \emptyset$.

A *maximal path* π in graph g is a list $\pi = v_1 v_2 \cdots v_{k+1}$ where for $1 \leq i \leq k$, $(v_i, s_i, v_{i+1}) \in E_g$, and $\text{in-deg}(v_1)=0$, $\text{out-deg}(v_{k+1})=0$, and $v_i = v_j \Rightarrow i = j$. In that case, $\text{word}(\pi) = s_1 s_2 \cdots s_k$.

A *cycle* π in graph g is a list $\pi = v_1 v_2 \cdots v_{k+1}$ where for $1 \leq i \leq k$, $(v_i, s_i, v_{i+1}) \in E_g$ and $v_1 = v_{k+1}$, and $v_i = v_j \Rightarrow i = j$ for $1 \leq i, j \leq k$. In that case, $\text{cword}(\pi) = \circ s_1 s_2 \cdots s_k$.

Let g_1 and g_2 be disjoint graphs over Σ and let $A_1 = \{a_{1,1}, \dots, a_{1,n}\} \subseteq V_{g_1}$, $A_2 = \{a_{2,1}, \dots, a_{2,m}\} \subseteq V_{g_2}$, $B_1 = \{b_{1,1}, \dots, b_{1,m}\} \subseteq V_{g_1}$, and $B_2 = \{b_{2,1}, \dots, b_{2,n}\} \subseteq V_{g_2}$ be disjoint ordered subsets. Then the *join* of g_1 and g_2 using A_1 , A_2 , B_1 , and B_2 is

$$g_1 \text{ }_{A_1, B_1 + A_2, B_2} g_2 = (V_{g_1} \cup V_{g_2}, E_{g_1} \cup E_{g_2} \cup \bigcup_{i=1}^n \{(A_{1,i}, \lambda, B_{2,i})\} \cup \bigcup_{i=1}^m \{(A_{2,i}, \lambda, B_{1,i})\}).$$

That is, we add unlabeled edges from nodes in A_1 and A_2 to the respective nodes in B_2 and B_1 .

5.2 String Tiles

A *port* (over the alphabet Φ) of graph g is a triple $p = (\sigma, I, O)$ where

- $\sigma \in \Phi \cup \bar{\Phi} \cup \{\lambda\}$ is called the *binding label*,
- $I \subseteq V$ is an ordered set of *input nodes* with $\text{in-deg } 0$,
- $O \subseteq V$ is an ordered set of *output nodes* with $\text{out-deg } 0$,
- $I \cap O = \emptyset$,
- $\sigma = \lambda$ iff $I = O = \emptyset$, in which case the port is said to be *empty*.

We use the notation σ_p , I_p , and O_p to refer to the binding label σ , input nodes I , and output nodes O of p , respectively. Two ports P_1 and P_2 are *disjoint* if I_{P_1} , O_{P_1} , I_{P_2} , and O_{P_2} are mutually disjoint.

A (k -sided) *string tile* over Σ, Φ is a pair $t = (P, G)$ where

- G is a directed graph over Σ s.t. all nodes have $\text{in-deg} \leq 1$ and $\text{out-deg} \leq 1$,
- $P = \{p_1, p_2, \dots, p_k\}$ is a set of mutually disjoint ports over Φ of G .

Here, Σ is the alphabet for string labels (and thus the alphabet for the language generated by the tiles) while Φ is the alphabet for the binding labels (which are relevant only for the self-assembly process). We use the notation P_t and G_t to refer to the ports P and graph G of t , respectively. When t is understood, σ_n , I_n^i and O_n^i refer to the binding label, the i^{th} input node and i^{th} output node of the port indexed by n , respectively. Two string tiles are *disjoint* if their graphs are disjoint. Note that the connected components of G are paths and cycles.

A string tile t is *of width* w if $|I_p| = |O_p| = w$ for all ports $p \in P_t$. A string tile is *uniform* if it is of width w for some w . A string tile t is *primitive*

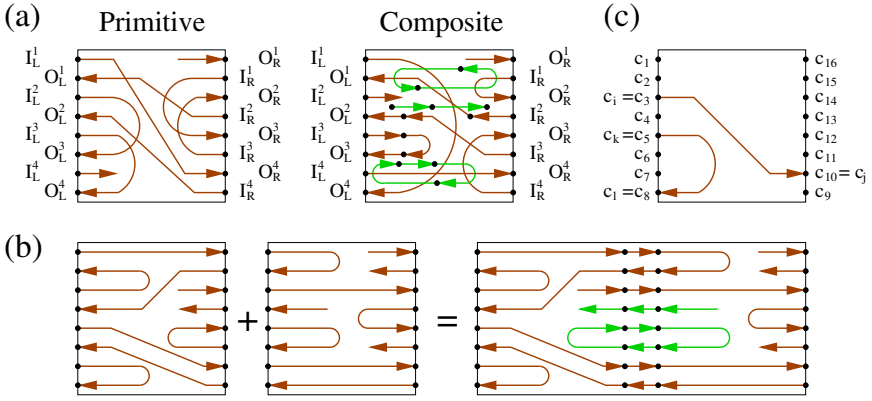


Fig. 11. (a) Diagram of a linear string tile (width 4). Note the order of the input and output nodes. The edge labels are not shown; nor are the binding labels. The half-edges indicate nodes that *aren't* involved in edges. (b) Illustration of the composition of tiles. (c) Numbering used in definition of planar tiles.

if $\bigcup_{p \in P_t} I_p \cup O_p = V_{G_t}$; i.e. every vertex is in a port. Note that in a primitive string tile, all edges are of the form (I_n^i, s, O_m^j) . If a tile is not primitive, it is *composite*. See Figure 11a.

Let t_1 and t_2 be disjoint string tiles over Σ, Φ and let p_1 and p_2 be ports of t_1 and t_2 respectively. Then we say that t_1 and t_2 are p_1 - p_2 -compatible if $\bar{\sigma}_{p_1} = \sigma_{p_2} \neq \lambda$, $|I_{p_1}| = |O_{p_2}|$, and $|O_{p_1}| = |I_{p_2}|$. For p_1 - p_2 -compatible tiles t_1 and t_2 , the p_1 - p_2 -composition is

$$t_1 \text{ }_{p_1+p_2} \text{ } t_2 = (P_{t_1} \cup P_{t_2} \setminus \{p_1, p_2\}, G_{t_1} \text{ }_{O_{p_1}, I_{p_1}} +_{O_{p_2}, I_{p_2}} G_{t_2}).$$

Note that it is easily verified that $t_1 \text{ }_{p_1+p_2} \text{ } t_2$ is indeed a string tile. If p_1 and p_2 are understood, as they are for all linear string tiles (below), then they are omitted as subscripts for notational convenience. See Figure 11b.

5.3 Classes of Tiles

A *linear* string tile is a 2-sided string tile such that one port (the *left* port, by convention indexed by L) is labelled from $\Phi \cup \{\lambda\}$ and the other port (the *right* port, indexed by R) is labelled from $\bar{\Phi} \cup \{\lambda\}$.

If t is a linear string tile, then it is called a *complete* tile if $\sigma_L = \lambda = \sigma_R$; it is called a *left cap* tile if $\sigma_L = \lambda \neq \sigma_R$; it is called a *right cap* tile if $\sigma_L \neq \lambda = \sigma_R$; and it is called an *internal* tile if $\sigma_L \neq \lambda \neq \sigma_R$. The intuition here is that ports with binding label λ cannot be extended.

A linear string tile with n input nodes of degree 0 and m output nodes of degree 0 is said to be n, m -nicked; if $n = m$, the tile is said to have n nicks. Note that a width w string tile always has $n = m$.

Let SET be a set of uniform linear string tiles, then SET^n is the subset of tiles with n or fewer nicks; SET_{lc} is the subset of left cap tiles; SET_{rc} is the subset of right cap tiles; SET_i is the subset of internal tiles. This notation is illustrated in Figure 8.

Let the *general tiles*, GEN , be the set of all uniform primitive linear string tiles. The *parallel tiles*, PAR , the *permutation tiles*, $PERM$, the *hairpin tiles*, $HAIR$, and the *planar tiles*, PLA , are subsets of GEN , defined as follows (and see Figures 8 and 11c). Let $t \in GEN$ be of width w . Then

- $t \in PAR$ iff $(I_n^i, s, O_m^j) \in E_{G_t} \Rightarrow i = j$ and $(n, m) \in \{(L, R), (R, L)\}$,
- $t \in PERM$ iff $(I_n^i, s, O_m^j) \in E_{G_t} \Rightarrow (n, m) \in \{(L, R), (R, L)\}$,
- $t \in HAIR$ iff $(I_n^i, s, O_m^j) \in E_{G_t} \Rightarrow$ either $i = j$ and $(n, m) \in \{(L, R), (R, L)\}$
or $i - j \in \{\pm 1\}$ and $(n, m) \in \{(L, L), (R, R)\}$,
- $t \in PLA$ iff for $i < k$, $(c_i, s, c_j), (c_k, s', c_l) \in E_{G_t} \Rightarrow$ either $i < k, l < j$ or $i, j < k, l$ or $l < i, j < k$, where $(c_1, c_2, \dots, c_{4W})$ is the counterclockwise list of port nodes $(I_L^1, O_L^1, I_L^2, O_L^2, \dots, I_R^2, O_R^2, I_R^1, O_R^1)$.

5.4 Assemblies and Languages

Let T be a finite set of width w linear string tiles. The string over T , $\alpha = t_1 t_2 t_3 \dots t_n$, with $n \geq 1$ and $t_i \in T$ for all $1 \leq i \leq n$, is an *assembly* over T if either $n = 1$ or t_i and t_{i+1} are compatible for all $1 \leq i < n$. Furthermore, α is a *maximal assembly* over T if α is not a proper substring of any other assembly over T . $\mathcal{A}(T)$ is the set of all maximal assemblies over T . Note that $\mathcal{A}(T)$ is always a regular language.

Each assembly α *induces* a single tile $t_\alpha = t'_1 + t'_2 + t'_3 + \dots + t'_n$, where t'_i is a unique isomorphic copy of t_i (required for distinctness). Note that for $n > 1$, this tile is always *not* a primitive tile!

A tile t (and thus an assembly α) *induces* a set of linear strings

$$L(t) = \{\text{word}(\pi) : \pi \text{ is a maximal path in } G_t\}$$

and a set of circular strings (only possible if the tile is composite)

$$C(t) = \{\text{cword}(\pi) : \pi \text{ is a cycle in } G_t\}.$$

Then, for T a finite set of linear string tiles, the linear and circular languages generated by T are, respectively,

$$L(T) = \bigcup_{\alpha \in \mathcal{A}(T)} L(t_\alpha)$$

and

$$C(T) = \bigcup_{\alpha \in \mathcal{A}(T)} C(t_\alpha).$$

Thus, we arrive at the family of languages generated by a (possibly infinite) set of linear string tiles, SET :

$$\mathcal{L}_{ST}(SET) = \{L(T) : T \subset SET \text{ is finite}\}$$

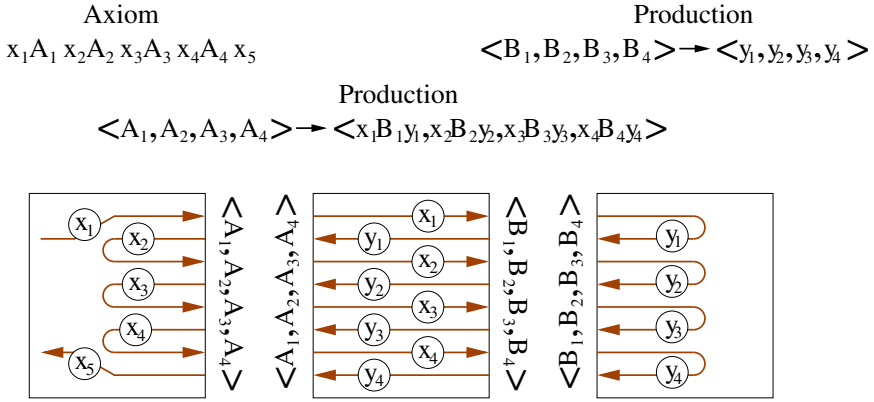


Fig. 12. The correspondence between a scattered grammar and parallel string tiles with $HAIR_{lc}^1$ and $HAIR_{rc}^0$.

and

$$\mathcal{C}_{ST}(SET) = \{C(T) : T \subset SET \text{ is finite } \}.$$

ST stands for “String Tile”. For sets of tiles $SET1$, $SET2$, and $SET3$, in the following we refer to the family of languages $\mathcal{L}_{ST}(SET1_{lc} \cup SET2_i \cup SET3_{rc})$ as $\mathcal{L}_{ST}(SET1, SET2, SET3)$, for notational convenience.

5.5 Scattered Linear Grammars for Parallel String Tiles

We will show that the family of languages generated by parallel tiles is a subclass of the languages generated by scattered context grammars. A *scattered context grammar* (see [RS97], page 128) is a quadruple $G = (\Sigma_N, \Sigma_T, P, S)$. Σ_N are the non-terminal symbols and Σ_T are the terminal symbols. S is a finite set of axiom strings over $\Sigma = \Sigma_N \cup \Sigma_T$ and P is a finite set of vector productions of the form

$$\langle A_1, A_2, \dots, A_n \rangle \rightarrow \langle x_1, x_2, \dots, x_n \rangle$$

where $A_i \in \Sigma_N$ and $x_i \in \Sigma^*$ and $n > 0$. A single derivation step, using the above production, transforms the string

$$\eta_1 A_1 \eta_2 A_2 \eta_3 \cdots \eta_n A_n \eta_{n+1} \rightarrow \eta_1 x_1 \eta_2 x_2 \eta_3 \cdots \eta_n x_n \eta_{n+1}$$

where $\eta_i \in \Sigma^*$. That is, we are performing synchronized application of context-free productions. The language generated by G is

$$L(G) = \{x : s \rightarrow^* x \text{ and } s \in S\}$$

where \rightarrow^* is the symmetric, transitive closure of \rightarrow .

We define a *scattered n -metalinear grammar* to be a scattered context grammar restricted to axiom strings of the form

$$x_1 A_1 x_2 A_2 x_3 \cdots x_n A_n x_{n+1}$$

and to productions of the form

$$\langle A_1, A_2, \dots, A_n \rangle \rightarrow \langle x_1 B_1 y_1, x_2 B_2 y_2, \dots, x_n B_n y_n \rangle$$

and

$$\langle B_1, B_2, \dots, B_n \rangle \rightarrow \langle y_1, y_2, \dots, y_n \rangle$$

where $A_i, B_i \in \Sigma_N$ and $x_i, y_i \in \Sigma_T^*$. That is, we are requiring synchronized application of linear productions to a string with n non-terminals. \mathcal{L}_{SM} is the family of languages generated by a scattered n -metalinear grammars, for some n .

Figure 12 shows a one-to-one correspondence of axioms to left cap tiles, productions to parallel tiles and right cap tiles, such that the language generated by the grammar is identical to the (linear) language generated by the tiles. In this way, it is straightforward to prove that

Theorem 1. $\mathcal{L}_{SM} = \mathcal{L}_{ST}(HAIR^1, PAR^0, HAIR^0)$.

5.6 Parallel Normal Form for Permutation Tiles

We will sketch the main ideas needed to prove that the languages generated by permutation tiles are the same as those generated by parallel tiles (although fewer permutation tiles may be required).

We use a two-step process for converting a finite set of permutation tiles (with general caps) into a finite set of parallel tiles (with hairpin caps) that generate the same language (Figure 13). Let $T_0 \subset GEN_{lc} \cup PERM_i \cup GEN_{rc}$; we will define $T_1 \subset GEN_{lc}^1 \cup PERM_i^0 \cup GEN_{rc}^0$ and $T_2 \subset HAIR_{lc}^1 \cup PAR_i^0 \cup HAIR_{rc}^0$, as sketched in Figure 13, by creating new maximal assemblies from the original ones, and collecting the new tiles to form T_1 and T_2 . Such an approach can easily guarantee that every string in the original language is still in the language generated by the new tiles; it must also be shown that no additional strings are generated.

Suppose without loss of generality that the tiles in T_0 are all width w , and that all maximal assemblies induce complete tiles (i.e., they have cap tiles on each end).

Step 1, removing nicks: For every assembly $\alpha \in \mathcal{A}(T_0)$, create a new assembly of width- $(w + 1)$ tiles for each maximal path in t_α . All the edges in the original tiles are present in the new tiles, but we keep only the edge labels on the chosen maximal path (which we imagine colored black); edge labels on the remaining edges (say, yellow) are replaced by λ . Additionally, we add λ -labelled edges from the left cap's bottom port to the start of the maximal path (red), and from the end of the maximal path back to the left caps' bottom port (cyan). All remaining

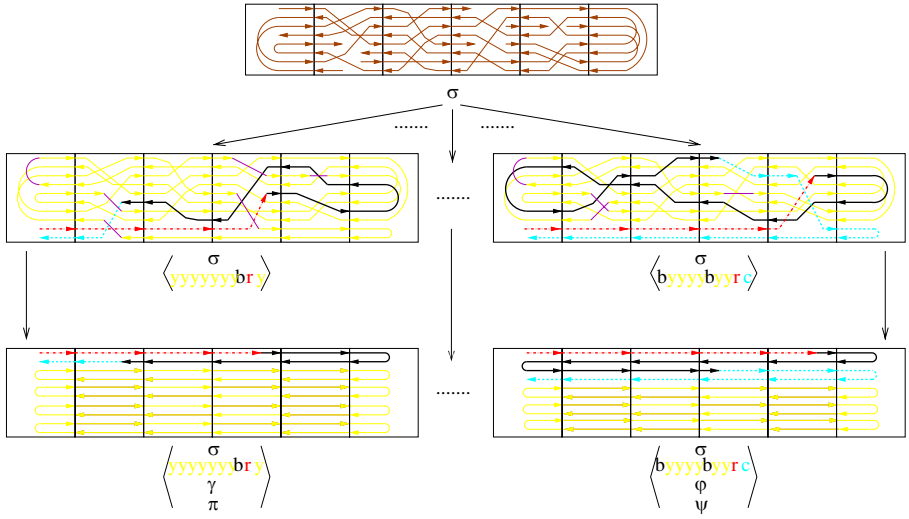


Fig. 13. The correspondence between permutation tiles and parallel tiles. At the top is an original assembly in $\mathcal{A}(T_0)$; in the middle are the corresponding assemblies in $\mathcal{A}(T_1)$, one assembly per maximal path in the original assembly; and at the bottom are the assemblies in $\mathcal{A}(T_2)$, where the chosen path has been moved to the upper rows. σ is the binding label joining the two central tiles in the original assembly. $\langle \sigma, yyyyyyybry \rangle$ is the binding label in step 1 augmented by the color pattern of the ports. γ is the permutation applied to the right port of the tile on the left, and π is applied to the left port of the tile on the right.

nicks are closed with λ -labelled edges (magenta). Finally, the binding labels are augmented by the color pattern of the ports. The union of all such tiles created for all maximal paths for all assemblies in $\mathcal{A}(T_0)$ is the set T_1 . Note that T_1 must be a finite set, because there are a finite number of possible width- $(w+1)$ tiles with the allowed edge and binding labels.

To see that $\mathcal{L}_{ST}(T_0) \subseteq \mathcal{L}_{ST}(T_1)$, note that for every word in $\mathcal{L}_{ST}(T_0)$ there is an assembly in $\mathcal{A}(T_0)$ that contains that word on a maximal path, and thus there is an assembly in $\mathcal{A}(T_1)$ that contains the same word on a maximal path.

To see that $\mathcal{L}_{ST}(T_1) \subseteq \mathcal{L}_{ST}(T_0)$, note that every assembly in $\mathcal{A}(T_1)$ induces a complete tile with a unique maximal path (and perhaps many circles). The augmented binding labels ensure that this path is colored red-black-cyan, and thus that it corresponds to a maximal path in the corresponding assembly in $\mathcal{A}(T_0)$ obtained by replacing each tile by one that had been used to create it.

Step 2, removing routing: For every assembly $\alpha \in \mathcal{A}(T_1)$, create a new assembly of width- $(w+1)$ tiles by permuting the input and output nodes on each original tile so that in the new assembly, all internal tiles are parallel tiles and the maximal path is layered at the top. Augment each port's binding label to include the permutations used for its input and output node lists. Finally, since

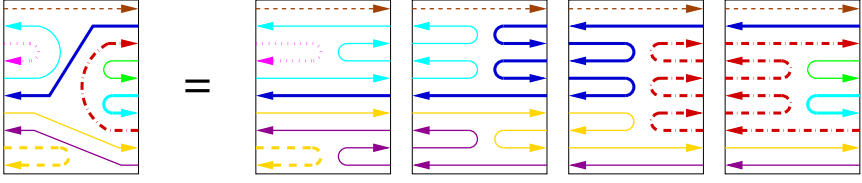


Fig. 14. Planar tiles can be built from hairpin tiles.

the yellow edges are all λ -labelled and involved in cycles, we replace the yellow edges in the cap tiles so as to obtain hairpin cap tiles. Again, the union of all such tiles created for all assemblies in $\mathcal{A}(T_1)$ is the set T_2 . Again, T_2 must be a finite set, because there are a finite number of possible tiles. The argument that $\mathcal{L}_{ST}(T_1) = \mathcal{L}_{ST}(T_2)$ is similar to the one given above.

Thus, we have sketched the proof for:

Theorem 2. $\mathcal{L}_{ST}(HAIR^1, PAR^0, HAIR^0) = \mathcal{L}_{ST}(GEN, PERM, GEN)$.

5.7 Hairpin Normal Form for General Tiles

A similar normal form, using hairpin tiles, can be found for general tiles. The argument has three steps. First, hairpin tiles are a normal form for planar tiles. Second, planar tiles are shown sufficient to generate the output languages produced by Hennie Machines. Third, a Hennie Machine can be found that outputs the language generated by any given set of linear string tiles.

Figure 14 illustrates the first step, which is given without proof:

Theorem 3. $\mathcal{L}_{ST}(HAIR^1, HAIR^0, HAIR^0) = \mathcal{L}_{ST}(HAIR^1, PLA^0, HAIR^0)$.

For the second step, we will use Turing Machines (TM) equipped with a 2-way read/write input tape and a 1-way write-only output tape. Let Q be the (finite) set of head states, Σ_I be the (finite) set of input tape symbols, and Σ_O be the (finite) set of output tape symbols. Then the state transition table for a TM consists of entries of the form $q\sigma \rightarrow q'\sigma'Ds$ where $q, q' \in Q$, $\sigma, \sigma' \in \Sigma_I$, $D \in \{L, R, H_A, H_R\}$, and $s \in \Sigma_O^*$; every pair in $Q \times \Sigma_I$ appears exactly once on the left-hand-side in the state transition table. The action H_A is to halt and accept; the action H_R is to halt and reject; L and R indicate moving left and right, respectively. A TM has the k -visit property, and hence is a *finite-visit* TM, if for no input does the machine enter any tape cell more than k times.

Related models include *Hennie Machines* (HM), which are finite-visit TMs whose use of tape space is bounded by a linear function of the input string length (i.e. they are linear bounded automata); and *two-way generalized sequential machines* (2gsm), which are Hennie Machines with a read-only input tape that is never accessed beyond the ends of the input string.

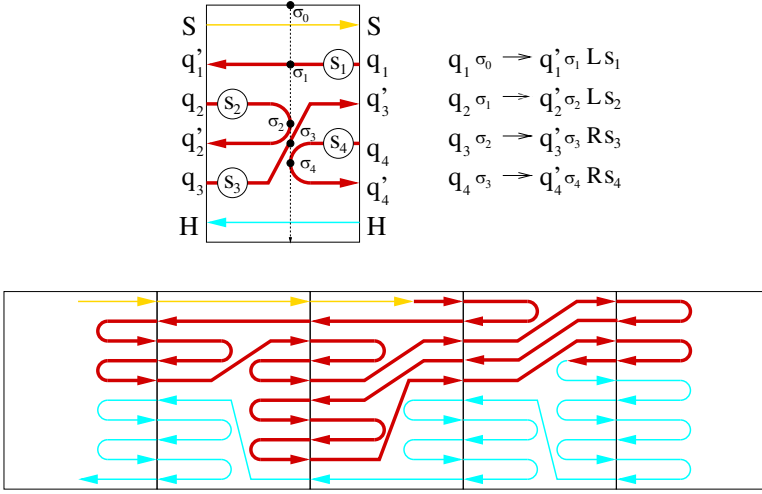


Fig. 15. Planar tiles that simulate a finite-visit Turing Machine. (top) The general form of tiles, showing start edges (yellow), action edges (red), and halt edges (cyan). Although formally not part of the tile, we label the cell with tape symbols σ_i along the center, indicating what must be on the tape during each visit of the Turing Machine head. Each action edge corresponds to an entry in the TM state transition table, as shown on the right. (bottom) An assembly containing the history of a Turing Machine computation.

If a TM m computing on input x enters the accepting halt state with y on the output tape, then we say that $y = m(x)$. The *output language* of a TM m is

$$L(m) = \{y : y = m(x) \text{ for some } x\}.$$

The family of all output languages of finite-visit TMs is

$$\mathcal{L}_{TM_{fin}} = \{L(m) : m \text{ is a finite-visit TM}\}.$$

Likewise,

$$\mathcal{L}_{HM} = \{L(m) : m \text{ is a HM}\}$$

and

$$\mathcal{L}_{2gsm_{fin}} = \{L(g) : g \text{ is a finite-visit } 2gsm\}.$$

By padding input strings with extra blank symbols, it is easy to see that $\mathcal{L}_{TM_{fin}} = \mathcal{L}_{HM}$. A corollary of our argument will be that, additionally, $\mathcal{L}_{TM_{fin}} = \mathcal{L}_{2gsm_{fin}}$.

Given a HM m , we must define a finite set of tiles $T \subset HAIR_{lc}^1 \cup PLA_i^0 \cup HAIR_{rc}^0$ that generate the same language; i.e. $L(m) = L_{ST}(T)$. For each $x \in \Sigma_I^*$, we draw the execution of m computing on x as shown in Figure 15, including a (imagine it yellow) path from the leftmost tile to the beginning of the input

where the head starts, and a path (cyan) from where the head halts back to the bottom of the leftmost tile, filling up all space in between. Each tile represents a single tape cell, and contains information about every visit to that cell by the Turing Machine head. The binding labels give the state of the Turing Machine head for each entry to and exit from the cell, thus ensuring that any maximal assembly represents a valid execution of the Turing Machine. The union of all tiles so created is T . Using a similar argument to that given for parallel normal form, we see that $L_{ST}(T) = L(m)$.

Theorem 4. $\mathcal{L}_{HM} \subseteq \mathcal{L}_{ST}(HAIR^1, PLA^0, HAIR^0)$.

For the third step of the argument, we need to find a Hennie Machine that outputs the same language as any given set of general tiles. Let $T \subset GEN$ be a finite set (without loss of generality, assume that all maximal assemblies of T have cap tiles on both ends) of width- w linear string tiles over Σ, Φ . Our HM will have input alphabet $\Sigma_I = \hat{T} = T \times \{L_i : 0 \leq i \leq w\} \times \{R_i : 0 \leq i \leq w\}$ and the output alphabet $\Sigma_O = \Sigma$. L_i and R_i are used only to select which path to read in assemblies containing multiple nicks. The HM proceeds in two phases: first it checks that the input represents a valid maximal assembly, then it reads the word off one of the maximal paths. The (finite) information about the tile types, edges, and labels is contained in the HM's finite state logic. The HM first checks that the first input symbol represents a left cap tile; then it scans right (producing no output) so long as each successive tile is compatible with the preceding one; either it arrives finally at a right cap tiles, or else it halts in the rejecting state, thus producing no output. In the former case, the HM then scans back to the left until it finds the first symbol (t, L_i, R_j) where O_L^i or O_R^j is nicked. (If it finds no such tile, it halts in the reject state.) The HM then simply follows the chosen edges from its beginning until its end, producing as output the edge labels, then halting in the accept state. Note that because symbols can contain L_0 and R_0 but ports are numbered starting from 1, the HM can copy any maximal path from any assembly. Also note that this HM uses the input tape for reads only, and hence is a finite-visit $2gsm$. Thus, we have:

Theorem 5. $\mathcal{L}_{ST}(GEN, GEN, GEN) \subseteq \mathcal{L}_{2gsm_{fin}}$.

Altogether,

Theorem 6. $\begin{aligned} \mathcal{L}_{ST}(HAIR^1, HAIR^0, HAIR^0) \\ &= \mathcal{L}_{ST}(GEN, GEN, GEN) \\ &= \mathcal{L}_{HM} = \mathcal{L}_{2gsm_{fin}}. \end{aligned}$

6 Conclusions and Open Questions

We can now fit the languages generated by linear string tiles into known language classes. *ETOL* systems [Roz73, RV78, RV80] are the most convenient well-studied model. Diagrams very similar to string tiles came up in the study of crossing sequences [Hen65] and transductions by finite-visit machines [EH98, EH99]. Of particular interest are the metalinear *ETOL* systems, which generate the languages

in $ETOL_{ml}$, and the $ETOL$ systems of finite index, which generate the language in $ETOL_{fin}$. It is straightforward to show that scattered metalinear grammars are a normal form for metalinear $ETOL$ systems, and thus $\mathcal{L}_{SM} = ETOL_{ml}$. In [ERS80] it was proved that $\mathcal{L}_{2gsm_{fin}} = ETOL_{fin}$. Furthermore, it was shown in [RV80] that the language $\{a^n b^n : n \geq 1\}^*$ is in $ETOL_{fin} \setminus ETOL_{ml}$. Indeed, there is a simple set of width-2 planar tiles that generate this language, and we can conclude that no set of parallel tiles can do so. Thus Figure 1 is justified.

We have given a full characterization only of the language classes generated by finite sets of tiles. However, complexity issues remain to be investigated – how many tiles are necessary to generate a specific language? The parallel normal form theorem potentially uses exponentially (in w) more parallel tiles than permutation tiles. This question has obvious relevance to using string tiles to solve NP-complete problems, such as CNF-SAT or DHPP.

Our definition of string tiles allows edges to be labeled by the empty string, corresponding to tiles with DNA containing no coding sequence. Thus, the resulting ligated DNA strands may have very long regions coding for no information. How do our language classes change if we insist on only λ -free string tiles?

Are the circular languages significantly different from the linear languages? Unlike linear DNA strands, circular DNA strands can be knotted with themselves and with other circular strands (although this is not part of the current formal model); can knottedness increase the computational power? Careful routing of strands using string tiles augments the computational power of linear DNA assembly; for 2D or 3D assembly, although string tiles cannot increase the language class beyond RE , can string tiles be used for more efficient computation?

Do the constructions and results presented in this paper point to better practical implementations for DNA-based computing? It is hard to say at this point, although the following calculation is illustrative. Consider a 40 variable CNF-SAT problem, with 160 clauses. In our construction, 80 string tiles must be prepared, each assembled from 240 prototiles (this number could be reduced substantially with an improved construction). These tiles would be 240×75 nm, with 160 sticky ends on each side. At a prototile concentration of $20\mu M$, one ml of solution would hold 12×10^{15} prototiles, thus 4×10^{13} tiles and 1×10^{12} maximal assemblies – just sufficient for 1X coverage of variable assignments. If self-assembly of these monsters were reliable (40 sticky-end sets would have to be sufficiently distinct) and roughly as fast as oligonucleotide hybridization, the maximal assemblies would form in a few minutes. The assembly containing the satisfying strand would still have to survive ligation at each of up to 160×40 nicks, at (on a good day) 90% yield for each nick. That doesn't leave much. On the one hand, we're excited to see a new approach for DNA based computing by self-assembly, which may have payoff for simple examples like generating an addition table; on the other hand, significant practical applications at this point seem rather far off.

Acknowledgements. The authors are indebted to Joost Engelfriet and Hendrik Jan Hoogeboom for their guidance through the maze of results on the output

languages of various sorts of transducers, and to John Reif and Thom LaBean for their critical reading, discussion, and encouragement.

References

- [Adl94] Leonard M. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266:1021–1024, November 11, 1994.
- [Adl00] Leonard M. Adleman. Toward a mathematical theory of self-assembly. USC Technical Report, 2000.
- [EH98] Joost Engelfriet and Hendrik Jan Hoogeboom. MSO definable string transductions and two-way finite state transducers. LIACS Technical Report 98-13, 1998.
- [EH99] Joost Engelfriet and Hendrik Jan Hoogeboom. Two-way finite state transducers and monadic second-order logic. In *Lecture Notes in Computer Science*, volume 1644, pages 311–320. Springer Verlag, 1999.
- [Eng99] Tony Eng. Linear DNA self-assembly with hairpins generates the equivalent of linear context-free grammars. In Rubin and Wood [RW99].
- [ERS80] J. Engelfriet, G. Rozenberg, and G Slutzki. Tree transducers, L systems, and two-way machines. *J. Comp. and Syst. Sc.*, 20:150–202, 1980.
- [FS93] Tsu-Ju Fu and Nadrian C. Seeman. DNA double-crossover molecules. *Biochemistry*, 32:3211–3220, 1993.
- [Hen65] H. C. Hennie. One-tape, off-line Turing machine computations. *Information and Control*, 8:553–578, 1965.
- [JKS98] Nataša Jonoska, Stephen A. Karl, and Masahico Saito. Three dimensional DNA structures in computing. In Lila Kari, Harvey Rubin, and David H. Wood, editors, *Proceedings of the 4th DIMACS Meeting on DNA Based Computers, held at the University of Pennsylvania, June 16-19, 1998*, pages 189–200, preliminary, 1998.
- [JKS99] Nataša Jonoska, Stephen A. Karl, and Masahico Saito. Creating 3-dimensional graph structures with DNA. In Rubin and Wood [RW99], pages 123–135.
- [Knu73] Donald E. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching (2nd ed)*. Addison-Wesley, 1973.
- [LL00] Michail G. Lagoudakis and Thomas H. LaBean. 2D DNA self-assembly for satisfiability. In Winfree and Gifford [WG00].
- [LSS99] Furong Liu, Ruojie Sha, and Nadrian C. Seeman. Modifying the surface features of two-dimensional DNA crystals. *Journal of the American Chemical Society*, 121(5):917–922, 1999.
- [LWR00] Thomas H. LaBean, Erik Winfree, and John H. Reif. Experimental progress in computation by self-assembly of DNA tilings. In Winfree and Gifford [WG00].
- [LYK⁺00] Thomas H. LaBean, Hao Yan, Jens Kopatsch, Furong Liu, Erik Winfree, John H. Reif, and Nadrian C. Seeman. Construction, analysis, ligation, and self-assembly of DNA triple crossover complexes. *Journal of the American Chemical Society*, 122:1848–1860, 2000.
- [MSS99] Chengde Mao, Weiqiong Sun, and Nadrian C. Seeman. Designed two-dimensional DNA Holliday junction arrays visualized by atomic force microscopy. *Journal of the American Chemical Society*, 121(23):5437–5443, 1999.

- [Rei99] John Reif. Local parallel biomolecular computing. In Rubin and Wood [RW99], pages 217–254.
- [Roz73] Grzegorz Rozenberg. Extension of tabled 0L-systems and languages. *Intern. J. Comp. Inform. Sci.*, 2:311–336, 1973.
- [RS97] Grzegorz Rozenberg and Arto Salomaa. *Handbook of formal languages*, volume 2. Springer-Verlag, New York, 1997.
- [RV78] G. Rozenberg and D. Vermeir. On ETOL systems of finite index. *Information and Control*, 38:103–133, 1978.
- [RV80] G. Rozenberg and D. Vermeir. On metalinear ETOL systems. *Fundamenta Informaticae*, pages 15–36, 1980.
- [RW99] Harvey Rubin and David Harlan Wood, editors. *DNA Based Computers III: DIMACS Workshop, June 23-25, 1997*, volume 48 of *DIMACS: Series in Discrete Mathematics and Theoretical Computer Science*, Providence, RI, 1999. American Mathematical Society.
- [See82] Nadrian C. Seeman. Nucleic-acid junctions and lattices. *Journal of Theoretical Biology*, 99(2):237–247, 1982.
- [See98] Nadrian C. Seeman. DNA nanotechnology: novel DNA constructions. *Annual Review of Biophysics and Biomolecular Structure*, 27:225–248, 1998.
- [SWY⁺98] N. C. Seeman, H. Wang, X. P. Yang, F. R. Liu, C. D. Mao, W. Q. Sun, L. Wenzler, Z. Y. Shen, R. J. Sha, H. Yan, M. H. Wong, P. Sa-Ardyen, B. Liu, H. X. Qiu, X. J. Li, J. Qi, S. M. Du, Y. W. Zhang, J. E. Mueller, T. J. Fu, Y. L. Wang, and J. H. Chen. New motifs in DNA nanotechnology. *Nanotechnology*, 9(3):257–273, 1998.
- [WG00] Erik Winfree and David K. Gifford, editors. *DNA Based Computers V: DIMACS Workshop, June 14-15, 1999*, volume 54 of *DIMACS: Series in Discrete Mathematics and Theoretical Computer Science*, Providence, RI, 2000. American Mathematical Society.
- [Win96] Erik Winfree. On the computational power of DNA annealing and ligation. In Richard J. Lipton and Eric B. Baum, editors, *DNA Based Computers: DIMACS Workshop, April 4, 1995*, volume 27, pages 199–221, Providence, RI, 1996. American Mathematical Society.
- [WLWS98] Erik Winfree, Furong Liu, Lisa A. Wenzler, and Nadrian C. Seeman. Design and self-assembly of two-dimensional DNA crystals. *Nature*, 394:539–544, 1998.
- [WYS98] Erik Winfree, Xiaoping Yang, and Nadrian C. Seeman. Universal computation via self-assembly of DNA: Some theory and experiments. In Laura F. Landweber and Eric B. Baum, editors, *DNA Based Computers II: DIMACS Workshop, June 10-12, 1996*, volume 44, Providence, RI, 1998. American Mathematical Society.

From Molecular Computing to Molecular Programming

Masami Hagiya

Graduate School of Science, University of Tokyo

hagiya@is.s.u-tokyo.ac.jp

Project home page: <http://hagi.is.s.u-tokyo.ac.jp/MCP/>

Abstract. The purpose of this article is to survey the research in molecular computing, including the achievements of the Japanese Molecular Computer Project, and foresee the future of the field. In addition to describing the major achievements of the project, Suyama's *Dynamic Programming Molecular Computer* and Sakamoto's *Hairpin Engines*, we summarize the computational paradigms related to molecular computing in order to provide a perspective on the field. We finally explain the idea of *molecular programming*, that the author is currently advocating.

1 Introduction

The Japanese Molecular Computer Project, sponsored by the Japan Society for the Promotion of Science, began in October 1996 and will end in March 2001 [8]. The major achievements of the project include Suyama's *Dynamic Programming Molecular Computer*, which drastically reduces the number of molecules in Adleman-style DNA computations and allows for automation by robots, and Sakamoto's *Hairpin Engines*, which employ hairpin forms, the typical secondary structures taken by DNA, for molecular computations. These achievements are supported by a series of theoretical studies by Yokomori's theory group, Nishikawa's simulator for DNA computations, Arita's recent work on code design, etc.

In particular, Sakamoto's two types of Hairpin Engine, *Whiplash PCR* and *SAT Engine*, are of great importance to the community of molecular computing, primarily because essential computational steps are implemented by self-directed hairpin formation, and no operations are necessary outside the test tube. Hairpin Engines thus realize a kind of *autonomous molecular computation*. This work also suggests the importance of *programming* molecules. In order to control successive autonomous reactions, including secondary structure formation, we should go beyond simple code design and carefully *program* DNA sequences. We believe that the technology of programming molecules should be one of the central research issues in the next decade. If it is ever established, it will be applied not only to computation, but also to many areas of molecular science.

This article surveys the research in molecular computing, including the above-mentioned achievements of our project, and foresees the future of the field.

Molecular computing aims to analyze the computational power of biomolecules such as DNA and protein, and seek engineering applications of the analyses. Engineering applications are not restricted to computers; they include any system that has information processing capabilities at the level of molecules. Information processing in nanomachines, for instance, should inevitably be implemented by molecular computation. Molecular biology, pharmacy, and medicine are also considered possible application areas. In particular, the application of molecular computation to biotechnology has recently been called *computationally inspired biotechnology*.

This article starts with Adleman-style DNA computing, which we call the *Adleman-Lipton Paradigm*, and efforts to improve the paradigm, including *Dynamic Programming Molecular Computer*. We then introduce the field of *autonomous molecular computing*, which tries to analyze computations by molecules in pure forms, and describe the achievements of the field, including *Hairpin Engines*. In order to provide a wider perspective of molecular computing, we also summarize the related computational paradigms. They include H-system, P-system, concurrency calculi, cellular computing, amorphous computing, etc. In the last section, the concept of *molecular programming*, which the author is currently advocating, is briefly described.

2 Improvements of the Adleman-Lipton Paradigm

In 1994, Adleman reported in *Science* a method of solving the Hamiltonian path problem using DNA molecules, and an experiment in which he actually solved the problem for a 7-vertex directed graph by the method [2]. Lipton generalized Adleman's method and proposed an approach to solving NP-complete problems by DNA molecules, in particular, the satisfiability problem of Boolean formulas (the SAT problem) [11]. In this article, we call this approach to DNA computing, proposed by Adleman and established by Lipton, the *Adleman-Lipton Paradigm*.

The Adleman-Lipton Paradigm solves combinatorial optimization problems, such as NP-complete problems, using DNA molecules. Each solution candidate of a problem is represented by a DNA molecule. The paradigm consists of the following two steps.

- (1) Solution candidates are randomly generated.
- (2) Real solutions are selected from among the generated candidates.

In the first step, the ability of DNA molecules to hybridize with one another through Watson-Crick base-pairing is exploited. In Adleman's experiment, vertices and edges in a directed graph are represented by single-stranded DNA molecules. They hybridize together to form a double-stranded DNA molecule that represents a path in the graph.

In the second step, real solutions are selected by molecular biology operations. Each operation is applied to all the candidates in a test tube. This step is a typical data-parallel computation.

The complexity of these molecular computations can be measured by the required time and the number of necessary molecules. Since the Adleman-Lipton paradigm reduces the time by sacrificing the number of molecules, the size of problems solvable by this method is limited by the number of molecules required for computations. So, the crucial point of the paradigm is how to reduce the number of potential candidates.

Many improvements upon the Adleman-Lipton paradigm have been proposed. Among others, Suyama et al. have proposed an iterative approach. Rather than generate all of the random candidates prior to selection, generation and selection of partial solutions is repeated [12]. In this approach, once candidates of partial solutions are identified, those that cannot be extended to a total solution are immediately removed. The remaining partial solutions are then extended, and the process is reapplied to these extended partial solutions.

Utilizing this approach, Suyama et al. designed an algorithm for solving the satisfiability problem of 3-literal clausal formulas (the 3-SAT problem) [25]. A similar algorithm had been independently proposed by Ogihara and Ray [14], but Suyama et al. actually implemented their algorithm using DNA molecules. The algorithm does not generate assignments of all the variables at once. Rather it extends partial assignments for one variable at a time. Those partial assignments that cannot be extended to complete satisfying assignments are removed.

Using this algorithm, they succeeded in solving a 4-variable 10-clause instance of the 3-SAT problem [25]. Currently, they are developing a DNA computing robot, shown in Fig. 1, with which they plan to solve a 30-variable 100-clause instance of the problem. According to Suyama's estimation, solving a 100-variable instance is not impossible. However, even if a 100-variable instance of the 3-SAT problem could be solved, DNA computers would not outperform electronic computers. Breakthroughs are required both in experimental technologies and in algorithms, before molecular computers based on the Adleman-Lipton paradigm can solve problems that are beyond the capability of electronic computers.

3 Autonomous Molecular Computing

In the first step of the Adleman-Lipton paradigm, solution candidates are generated by the ability of DNA molecules to hybridize together. In this step, the computation proceeds without any operations outside the test tube. On the other hand, in the second step, a large number of laboratory operations are applied from outside the tube.

Molecular computations are called *autonomous* if they proceed by successive autonomous reactions of molecules. The first step of the Adleman-Lipton paradigm is a typical example of autonomous molecular computation. They are also called *one-pot reactions*, because they autonomously proceed in a single test tube once the necessary ingredients are put into the tube initially. Research on autonomous molecular computation is considered to analyze the computational power of molecules in pure forms. In the rest of this section, we describe Sakamoto's research after briefly touching upon Winfree's.

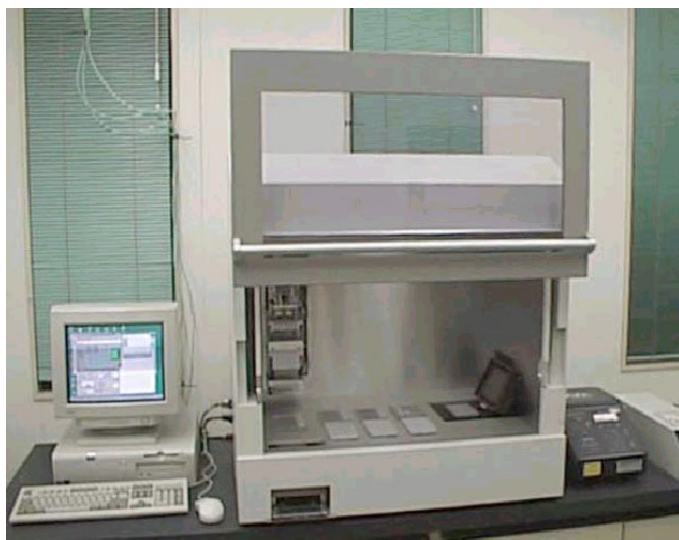


Fig. 1. Suyama's robot.

3.1 Winfree's DNA Tiles

Hybridization of DNA molecules is also called *self-assembly*, because DNA molecules autonomously assemble together. The computational power of self-assembly by DNA molecules has been thoroughly investigated by Winfree [23]. In particular, he pointed out the computational power of DNA molecules called *DNA tiles*. *Double-crossover molecules* are typical examples of DNA tiles (Fig. 2, lower left). These are structures made of two double-stranded molecules which exchange their single strands at two points. Each double-crossover molecule is composed of four single strands that self-assemble.

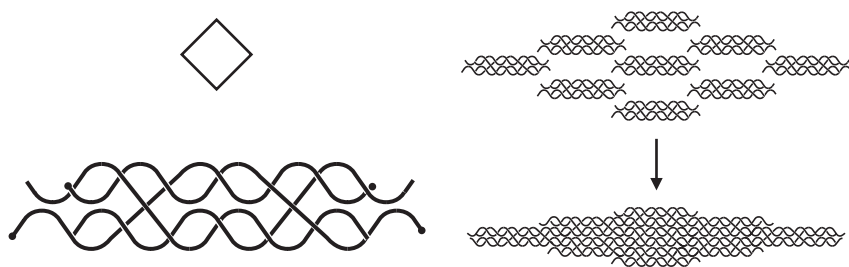


Fig. 2. Winfree's DNA tiles.

Double-crossover molecules can also self-assemble together and form a planar structure [22], because they have four sticky ends, each hybridizing with a sticky end of another molecule (Fig. 2 right). This process of self-assembly corresponds to that of square tiles that have colored edges and can hybridize only if adjacent edges are of the same color (Fig. 2, upper left).

The tiling process of square tiles with colored edges is known to have high computational power [23]. For instance, it can simulate execution of one-dimensional cellular automata, and thus has universal computability.

3.2 SAT Engine

Sakamoto and his group, which includes the author, have been investigating computational schemes employing hairpin forms of DNA molecules. So far, they have proposed two computational schemes and have finished preliminary experiments for their implementation. Hairpin forms are the most fundamental of the secondary structures taken by DNA molecules. In this sense, the work of Sakamoto et al. is considered the first step towards establishing computations employing molecular structures. Sakamoto calls these computational schemes based on hairpin formation *Hairpin Engines*.

SAT Engine, recently reported in *Science*, is a computational scheme for solving the satisfiability problem of clausal formulas [19]. Since a clausal formula is a conjunction of clauses, to satisfy such a formula is to satisfy all of its clauses. Since a clause is a disjunction of literals, to satisfy a clause is to satisfy at least one of its literals. Therefore, in order to satisfy a clausal formula, it suffices to select one literal from each clause and compose an assignment by the selected literals. For instance, if literal a is selected from the clause, $(a \vee \neg b \vee c)$, then a is made true in the composed assignment. If $\neg b$ is selected, b is made false. If a variable and its negation are both selected (from different clauses), it is impossible to compose an assignment since they are inconsistent. On the other hand, if inconsistent literals are not selected, one can obtain a satisfying assignment.

SAT Engine is based on the idea to represent complementary literals, i.e., a variable and its negation, by Watson-Crick complementary sequences of DNA. The literals selected from the clauses are concatenated into one single-stranded DNA molecule. If the selection contains inconsistent literals, the molecule contains complementary sequences and forms a hairpin structure (Fig. 3). If the selection is consistent, the molecule does not form a hairpin structure. Therefore, it is possible to decide whether a clausal formula is satisfiable by generating a random pool of molecules, each composed of literals selected from clauses, and distinguishing the molecules with hairpins from those without hairpins.

SAT Engine uses the following two methods to distinguish hairpins from non-hairpins.

- (1) The recognition site of a restriction enzyme is inserted into the middle of a sequence encoding a literal. Molecules with hairpins are cut by the enzyme.
- (2) The efficiency of amplification by PCR differs for molecules with hairpins and without hairpins. The former kind of molecule is less efficient in amplification than the latter. Sakamoto developed a new kind of PCR, called

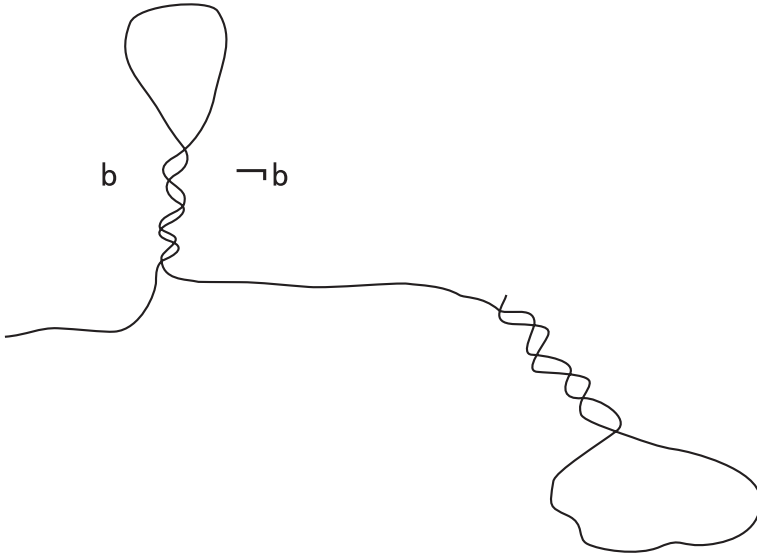


Fig. 3. Inconsistent assignment.

exclusive PCR (ePCR), which exaggerates the difference between the two kinds of molecules.

Using the above two methods, molecules with hairpins and molecules without hairpins can be separated. Although this process does require a number of laboratory operations, the most important part of the computation, the detection of inconsistent assignments, is realized by hairpin formation, which is an autonomous reaction of the molecules. Note also that the number of laboratory operations is not dependent on the number of variables or clauses. Sakamoto et al. actually solved a 6-variable 10-variable instance of the 3-SAT problem [19].

3.3 Whiplash PCR

Whiplash PCR is a method to realize state machines using DNA molecules [7]. Sakamoto et al. have been trying to implement state machines that have internal programs, using DNA molecules. Remember that the Adleman-Lipton Paradigm is a computational scheme for data-parallel computations, in which programs are realized as sequences of laboratory operations. In contrast to the Adleman-Lipton Paradigm, Sakamoto et al. tried to represent programs by the molecules themselves. If each molecule has its own program, program-parallel computations can be implemented.

Whiplash PCR is based on the idea that the 3'-end of a single-stranded DNA molecule encodes the current state of the machine implemented by the

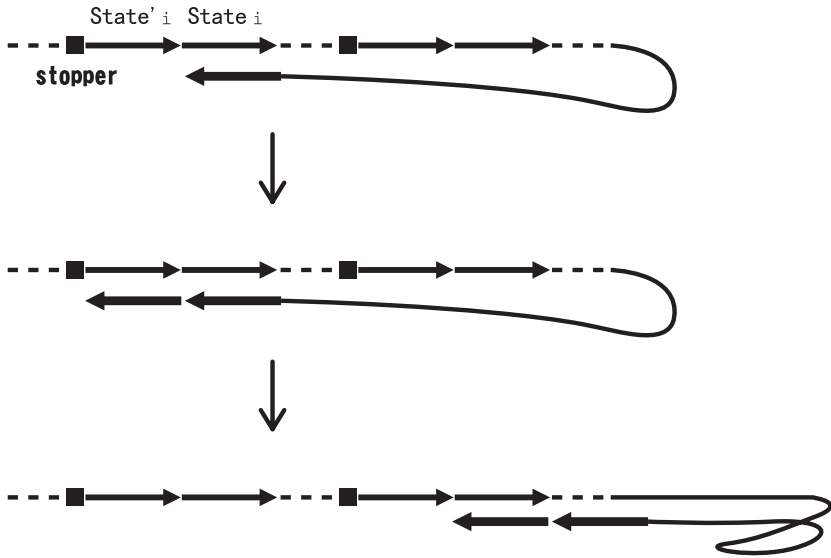


Fig. 4. Whiplash PCR.

molecule. It also contains a table for state transitions, which is the program of the machine. As in Fig. 4, the state transition table is a repetition of sequences, each of which is a concatenation of the stopper sequence (**stopper**), the state before the transition (State'_i) and the state after the transition (State_i).

By taking a hairpin form, the 3'-end hybridizes with some before-state (State_i) in the transition table of that molecule. This molecule is extended by the DNA polymerase. As a consequence, the complement of the after-state (State'_i) is attached to the 3'-end. This means that the current state is changed from State_i to State'_i . As in the figure, if the hairpin structure is denatured and another hairpin is formed, the next transition can occur. It is reported that several (about five) transitions have been experimentally confirmed [10].

The stopper sequence leads to cessation of extension by the DNA polymerase. There are many methods of implementing the stopper sequence. One that has been implemented is to have only three of the four possible deoxyribonucleotide triphosphates in the reaction buffer, and represent the stopper sequence as a polynucleotide of the base that is complementary to the one that is absent from the reaction mixture.

By appropriately designing transition tables, it is possible to implement many kinds of computation by Whiplash PCR. Sakamoto et al. proposed a computational scheme in which both a program that evaluates a μ -formula (read-once Boolean formula) and an input to the program are put on a single DNA molecule. This molecule then behaves as an independent computational unit and computes the value of the formula as a function of the input [7].

4 New Computational Paradigms

Around molecular computing, a number of new computational paradigms are emerging. These include:

- DNA computing
- molecular computing
- chemical computing
- aqueous computing
- crystal computing
- cell computing
- gel computing
- amorphous computing

These computational paradigms can be classified within the context of molecular computing into four kinds of computation:

- (1) computing inside a single molecule
- (2) computing by interactions among molecules
- (3) computing with membranes
- (4) computing with geometry

Note that each group is a source of great computational power. Also, there are connections between these groups.

4.1 Computing Inside a Single Molecule

Hairpin Engines are typical examples of computation inside a single molecule. In both SAT Engine and Whiplash PCR, computation is implemented by conformational change or structural formation of a DNA molecule.

Protein folding is also a typical example of structural formation by a single molecule. Fraenkel was the first to explicitly state the relationship between protein folding and computation [6]. He showed that protein folding is NP-complete by reducing the energy minimization problem of spin glasses, which was known to be NP-complete, into the protein folding problem. This result, however, is purely theoretical and there is a big gap between Fraenkel's model and an actual protein.

There are some proposals to implement nano-scale memory by partial modification of a single molecule. Although such proposals do not provide universal computational schemes, they would become the first practical molecular computers. In the *Stickers Model* of Roweis et al., read/write memory is implemented by attaching short DNA molecules, called *stickers*, to a long single-stranded DNA molecule [17].

Head and Yamamura showed that even write-once memory has high computational power [9]. They proposed *Aqueous Computing*, in which write-once memory is used to solve NP-complete problems such as the max clique problem. In their current approach, write-once memory is implemented by a plasmid

containing sequences called *stations*, each corresponding to one bit. A station is surrounded by the recognition sites of the same restriction enzyme, and writing on a station is implemented by removing the station from the plasmid through sequential restriction endonuclease digestion and ligation.

4.2 Computing by Interactions among Molecules

Typical examples of computation by molecular interactions are those by hybridization of DNA molecules. Almost all the computational schemes proposed in the field of DNA computing employ DNA hybridization. Hybridization of DNA is also called self-assembly, because DNA molecules autonomously assemble together.

Winfree characterized the computational power of self-assembly for various forms of DNA molecules [23]. He showed that the set of double-stranded molecules that can be formed by self-assembly from a finite set of single-stranded molecules is a regular language. If branching is allowed in double-stranded molecules, the power of self-assembly is increased to that of context-free languages, because branching molecules can represent structures corresponding to the derivation trees of a context-free grammar. Furthermore, the self-assembly of DNA tiles is universally computable (Section 3.1).

The *splicing system* introduced by Head, also known as the *H-system*, is a formal model of recombination of DNA, i.e., cutting by restriction enzymes and concatenation by ligase [16]. Long before Adleman's work, Head had modeled reactions of restriction enzymes and ligase, which are now employed in many computational schemes of DNA computing. So far, a huge number of theoretical studies on the H-system have been done from the perspective of formal language theory.

Abstract chemistry is a field in *artificial life* that investigates molecular interactions in terms of artificial chemical systems. A large number of abstract chemical systems have been proposed and analyzed by simulations (e.g., [4]). Studies on cellular automata can also be classified in the same research direction.

Computational schemes involving combinations of self-assembly and conformational change of molecules have also been proposed. It is known that by allowing molecules to change their structures, the computational power of self-assembly increases [24,18].

4.3 Computing with Membranes

Membranes are the most basic device for dividing solutions into separate partitions. Since solutes cannot move across a membrane, independent reactions can occur in each partition of the solution. A partition divided by a membrane is called a *compartment*.

Several methods for implementing membranes are known. *Mechanical-electrical micro-systems* (MEMS), or chemical IC, are small chemical plants

implemented on a chip consisting of small reaction chambers, each considered to be a compartment. Liposomes are small spheres made of lipid bilayers that separate solutions into an inside and outside. There are some proposals for computing with liposomes, but their implementation is not easy.

A cell is the most typically encountered compartment. Many ideas have been proposed for computing with living cells. Reactions in a cell that can be employed for computation include metabolism, gene regulation and signal transduction. Creation of cell machines by artificially regulating such reactions has been proposed [21].

Many formal models have been proposed for computing with membranes. Concurrency calculi, such as *chemical abstract machines* [5], are examples of such models. So far, many calculi, including π -calculus, join calculus, and ambient calculus, have been proposed, but they are not strongly related to actual chemical reactions.

The *P-system*, proposed by Păun, is also a formal model for computing with membranes [15]. Strings are rewritten inside a membrane. Deletion of a membrane is also regulated by a rewrite rule. Like the H-system, the P-system has been investigated from the perspective of formal language theory.

4.4 Computing with Geometry

Computations with a membrane employ the very simple topology consisting of the inside and outside of the membrane. Computations with a richer topology, including 2-D or 3-D geometry, are worth investigating. For instance, each DNA tile in the entire structure has a 2-D or 3-D coordinate. Computations employing the coordinate of each tile would be very powerful.

Amorphous computing, proposed by Abelson et al., is a computational paradigm with computational particles scattered in a 2-D or 3-D field in an amorphous fashion [1]. Since it does not assume regular crystal structures, it would have many applications.

Each computational particle can emit transmission substances that diffuse and reach another particle, which senses the substances, changes its internal state, and emits its own transmission substances. Abelson et al. developed algorithms for computational particles to acquire positional information and create patterns. They even developed a programming language for describing the patterns. To implement amorphous computing, they propose *cellular computing* [17].

Amorphous computing is independent from molecular computing, but there are strong relationships between the two ideas. For example, it would be possible to implement amorphous computing with DNA tiles. Gel can also be employed for amorphous computing.

5 Molecular Programming

In molecular computing, engineering viewpoints are considered very important, because they distinguish molecular computing from related research fields such

as mathematical biology, complex systems, artificial life, etc. The ultimate goal of molecular computing should be to synthesize artificial information processing systems at the molecular level. To achieve this goal, new computational paradigms, new algorithms, and new programming languages need to be developed. We use the phrase, *molecular programming*, to explicitly denote research into this synthetic approach.

Molecular programming is a field that aims to *program* reactions of biomolecules. Programs that control reactions of biomolecules are classified into those that are encoded in molecules themselves, and those that are implemented as sequences of laboratory operations. The former kind of program is characteristic to biomolecules that have combinatorial complexity and autonomous computational power. By coordinating the former and latter kinds of programs, it should be possible to control reactions of biomolecules and thus realize molecules or molecular systems with the intended functions and structures.

In a sense, molecular programming is everywhere. In PCR, for instance, it is important to design good primer sequences, and this is an example of the first kind of programming. In addition, choosing appropriate reaction conditions, such as temperature, salt concentration, etc., is also important, and is the second kind of programming. Only by coordinating the two programs, it is possible to have efficient amplification.

Another example of the first kind of programming is the code design of DNA sequences in molecular computing. This problem is related to the design of PCR primers, and it has been actively investigated in the field of molecular computing (e.g., [3]). As the second kind of programming, we can classify research on modeling and simulating chemical reactions (e.g., [13]).

A more sophisticated example of molecular programming is that of DNA chips. Suyama has been developing a universal DNA chip [20], where mRNA transcripts are first converted to standard DNA sequences, which Suyama calls *DNA coded numbers* (DCN's), and the DCN's are then hybridized with oligomers attached to the DNA chip (Fig. 5). The process to convert mRNA transcripts to DCN's employs template sequences that hybridize with both the mRNA transcripts and the DCN's.

By changing the set of template sequences, it is possible to change the conversion and thus change the set of mRNA that can be detected by the chip. In this sense, the chip is universal.

Based on the universal chip, we can further develop a more intelligent DNA chip. Rules for genetic diagnoses are in general of the following form:

If gene A is expressed, gene B is not expressed and gene C is expressed,
then there is a danger of disease D.

This kind of rule can be naturally represented by a Boolean formula (or decision diagram). Since a large number of methods for evaluating Boolean formulas using molecules have been proposed, including Whiplash PCR, it is possible to make such diagnoses by DNA computation.

In Suyama's universal chip, mRNA transcripts are converted to DCN's. Therefore, DCN's can be directly used for computation. Using Whiplash PCR,

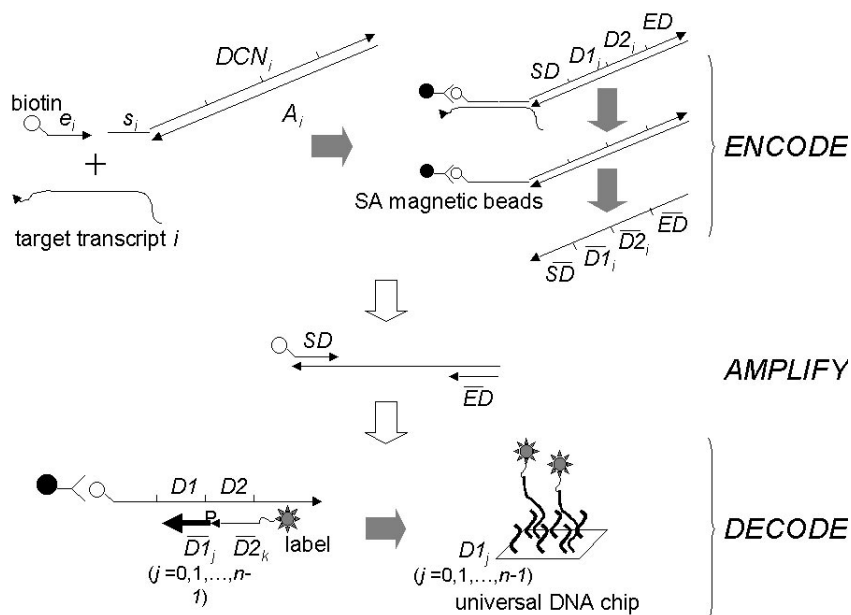


Fig. 5. Suyama's universal DNA chip.

even rules for diagnoses can be represented by molecules, so all the information processing can be done in a test tube.

The advantages of this intelligent DNA chip are:

- The mRNA transcripts need not be sequenced.
- Less information is revealed using a digital form. The data on mRNA transcripts are not revealed. In some cases, the rules for diagnoses can also be kept secret.

The above-mentioned intelligent DNA chips are one of the anticipated applications of molecular programming. Because of its fundamental nature, the application areas of molecular programming are considered broad and to cover various fields related to biomolecules, including genetic analysis, nanotechnology, nanomachine engineering, and combinatorial chemistry.

Acknowledgements. We deeply thank all the members of the Japanese Molecular Computer Project, which is supported by the Japan Society for the Promotion of Science under grand JSPS-RFTF 96I00101.

References

1. Harold Abelson, Don Allen, Daniel Coore, Chris Hanson, George Homsy, Thomas F. Knight, Jr., Radhika Nagpal, Erik Rauch, Gerald Jay Sussman, and Ron Weiss: Amorphous Computing, *Communications of the ACM*, Vol.43, No.5, pp.74–82, 2000.
2. Leonard M. Adleman: Molecular Computation of Solutions to Combinatorial Problems, *Science*, Vol.266, pp.1021–1024, 1994.
3. Masanori Arita, Akio Nishikawa and Masami Hagiya: Improving Sequence Design for DNA Computing, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2000*, July 10–12, Las Vegas, Nevada, pp.875–882, 2000.
4. Wolfgang Banzhaf, Peter Dittrich and Burkart Eller: Selforganization in a system of binary strings with topological interactions, *Physica D*, Vol.125, pp.85–104, 1999.
5. Gérard Berry and Gérard Boudol: The Chemical Abstract Machine, *the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 1990, pp.81–94. *Theoretical Computer Science*, Vol.96, 1992, pp.217–248.
6. Aviezri S. Fraenkel: Protein Folding, Spin Glass and Computational Complexity, *DNA Based Computers III, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, Vol.48, pp.101–121, 1999.
7. Masami Hagiya, Masanori Arita, Daisuke Kiga, Kensaku Sakamoto and Shigeyuki Yokoyama: Towards Parallel Evaluation and Learning of Boolean μ -Formulas with Molecules, *DNA Based Computers III, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, Vol.48, pp.57–72, 1999.
8. Masami Hagiya: Perspectives on Molecular Computing, *New Generation Computing*, Vol.17, No.2, pp.131–140, 1999.
9. Tom Head, Masayuki Yamamura and Susannah Gal: Aqueous Computing: Writing on Molecules, *Congress on Evolutionary Computation*, July 6–9, 1999, Mayflower Hotel, Washington D.C., USA, pp.1006–1010, 1999.
10. Ken Komiya, Kensaku Sakamoto, Hidetaka Gouzu, Shigeyuki Yokoyama, Masanori Arita, Akio Nishikawa and Masami Hagiya: Successive State Transitions with I/O Interface by Molecules, *DNA6, Sixth International Meeting on DNA Based Computers*, Leiden Center for Natural Computing, June 13–17, 2000, pp.1–30, 2000.
11. Richard J. Lipton: DNA Solution of Hard Computational Problems, *Science*, Vol.268, pp.542–545, 1995.
12. Nobuhiko Morimoto, Masanori Arita and Akira Suyama: Solid Phase DNA Solution to the Hamiltonian Path Problem, *DNA Based Computers III, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, Vol.48, pp.193–206, 1999.
13. Akio Nishikawa, Masayuki Yamamura and Masami Hagiya: DNA Computation Simulator Based on Abstract Bases, *Soft Computing*, to appear, 2000.
14. Mitsunori Ogihara and Animesh Ray: DNA-Based Parallel Computation by “Counting”, *DNA Based Computers III, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, Vol.48, pp.255–264, 1999.
15. Gheorghe Păun: Computing with Membranes, *TUCS Research Report*, No.208, November 1998, www.tucs.fi.
16. G. Păun, G. Rozenberg, and A. Salomaa: *DNA Computing*, Springer, 1998.
17. Sam Roweis, Erik Winfree, Richard Burgoyne, Nickolas V. Chelyapov, Myron F. Goodman, Paul W. K. Rothmund, and Leonard M. Adleman: A Sticker Based Model for DNA Computation, *DNA Based Computers II, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, Vol.44, pp.1–29, 1999.

18. Kazuhiro Saitou: Self-Assembling Automata: A Model of Conformational Self-Assembly, *Pacific Symposium on Biocomputing'98*, pp.609–620, 1998.
19. Kensaku Sakamoto, Hidetaka Gouzu, Ken komiya, Daisuke Kiga, Shigeyuki Yokoyama, Takashi Yokomori and Masami Hagiya: Molecular Computation by DNA Hairpin Formation, *Science*, Vol.288, pp.1223–1226, 2000.
20. Akira Suyama, Nao Nishida, Ken-ichi Kurata, Katsumi Omagari: Gene Expression Analysis by DNA Computing, *Currents in Computational Molecular Biology* (ISBN 4-946443-61-4), pp.12–13, 2000.
21. Ron Weiss and Thomas F. Knight, Jr.: Engineered Communications for Microbial Robotics, *DNA6, Sixth International Meeting on DNA Based Computers*, Leiden Center for Natural Computing, June 13–17, 2000, pp.5–19, 2000.
22. Erik Winfree, Furong Liu, Lisa A. Wenzler, and Nadrian C. Seeman: Design and self-assembly of two-dimensional DNA crystals, *Nature*, Vol.394, pp.539–544, 1998.
23. Erik Winfree, Xiaoping Yang and Nadrian C. Seeman: Universal Computation via Self-assembly of DNA: Some Theory and Experiments, *DNA Based Computers II, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, Vol.44, pp.191–213, 1999.
24. Takashi Yokomori: YAC: Yet Another Computation Model of Self-Assembly, *Preliminary Proceedings of the Fifth International Meeting on DNA Based Computers*, June 14–15, 1999, Massachusetts Institute of Technology, pp.153–167, 1999.
25. Hiroshi Yoshida and Akira Suyama: Solutions to 3-SAT by Breadth First Search, *Preliminary Proceedings of the Fifth International Meeting on DNA Based Computers*, June 14–15, 1999, Massachusetts Institute of Technology, pp.9–20, 1999.

Graph Replacement Chemistry for DNA Processing

John S. McCaskill¹ and Ulrich Niemann²

¹GMD-National Research Center for Information Technology
Schloß Birlinghoven, St. Augustin 53754, Germany

²Institut für Molekulare Biotechnologie,
Beutenbergstr. 11, D-07745 Jena
email: mccaskill@gmd.de

Abstract. The processing of nucleic acids is abstracted using operators on directed and labeled graphs. This provides a computational framework for predicting complex libraries of DNA/RNA arising from sequences of reactions involving hybridisation intermediates with significant combinatorial complexity. It also provides a detailed functional classification scheme for the reactions and side-reactions of DNA processing enzymes. It is complementary to the conventional string-based DNA Computing grammars such as splicing systems, in that the graph-based structure of enzyme-nucleic acid complexes is the fundamental object of combinatorial manipulation and in that the allowed reactions are specified by local graph replacement operators (i.e. catalysts for structural transitions) associated with enzymes. The focus of the work is to present a calculus for the compact specification and evaluation of the combined action of multiple DNA-processing reactions. Each enzyme and its side-reactions may be classified by a small set of small graph replacement operators. Complex replication and computation schemes may be computed with the formalism.

1. Introduction

This work describes a formalism for dealing with the combinatorial complexity of enzymatically-catalysed reactions in nucleic acid complexes, which makes them useful for complex diagnostics and DNA Computing. The general framework may be useful for other domains of combinatorial chemistry such as protein or carbohydrate chemistry. Chemical reactions are introduced as families of local term replacements (productions) in a graph replacement system. As such, the work is in the tradition of the string rewriting systems introduced by Thue^[1], which has been extended to the graph^[2] and term^[3] rewriting systems which with the concept of reduction have played an important role in the modern theory of programming languages. Apart from the application to molecular computation, the current work differs from the conventional focus of this tradition in two respects. Chemical structures are seen not only as the terms themselves but also as encoding the rewriting rules which act on chemical structures. Secondly, in DNA processing it makes a major difference whether a molecule is reconverted to itself or creates a copy of itself (these are not distinguished in the "word problem" of string rewriting systems).

The cyclical manipulations of DNA giving rise to isothermal amplification are useful but rare properties of DNA processing systems which can be computed by the current approach. Formal artificial chemistries have been studied by many authors, culminating in the lambda calculus chemistry of Fontana^[4]. The current work seeks to complement these studies in a more practical way, by staying close to the known structural and catalytic mechanisms of DNA processing and providing a bridge to the experimental design of combinatorially complex DNA interconversion and amplification systems.

We briefly outline the major features of our polynucleotide graph processing formalism. Both bimolecular recognition and unimolecular processing reactions can be specified compactly by extending standard graph replacements to include variables (both as edges and labels). The analysis of new amplification schemes requires both the distinction between multiple copies of a structure (i.e. multi-sets), and reactions which generate multiple product complexes. Multi-step reactions are decomposed into their component reactions, allowing iterative reactions such as polymerisation to be described compactly; for example in terms of initiation, elongation and termination. Multiple product complexes (e.g. cleavage) are allowed. Catalysts and their translocation in the course of reaction may be treated explicitly or implicitly. Such a compact specification of reactions and catalysis is useful both for an investigation of side-reactions in experiments involving reactions among nucleic acids, such as amplification cycles and for DNA Computing, and for an investigation of minimal catalytic sets for generating combinatorial product families.

The basic concepts of molecular graphs are introduced in section 2 followed by the extension of graph replacement systems to molecular graph replacements in sections 3 and 4. A computer program MOLGRAPH (coded in C) which implements this calculus is described in section 5. As an example, the application to an isothermal amplification scheme based on the 3SRreaction^[9] is described in section 6.

2. Molecular Graphs

A graph is composed of a set of vertices and a set of edges connecting two vertices^[5]. A molecule consists of atoms and bonds between them. Their representation by graphs using the one letter labelling of atoms and the single and double bond labelling of bonds is standard. In the description of chemical reactions it is also common practice to lump together functional groups (e.g. OH, COOH, CH₃) and indeed whole unaltered segments of molecules (e.g. R) as single nodes. For larger fragments as nodes, there arises the problem of specifying the attachment point of edges to the internal structure in the node. In the case of DNA, the two edges binding the monomer into the backbone chain are labelled by their point of attachment (5' or 3' carbon atom) to the ribose sugar ring. For the description of classes of reactions it will be necessary to extend the usual labelling of graphs in two ways. Firstly each node and edge label is divided up into a list of attribute-value pairs. Secondly each attribute value is ascribed a property: *variable* or *constant*. Classes of graphs can be specified by such *variable graphs*.

Definition 1: A *variable graph* is a vertex and edge labelled graph with a labelling set $\Sigma_{VE} = A \times VAR$, where A is a set of *attribute values* and VAR is a set of *attribute properties in the form of strings* representing *variable names* including the special strings *don't care* and *match*.

Variable graphs are introduced to allow more general matching between graphs. The matching may consider not only the structure of the graph, but also attribute values of a vertex or edge. The mutually exclusive flags - *don't care*, *match* or a *variable name* - indicate how the corresponding attribute value will be treated in graph matching: ignored, respected or a variable name. If a variable name is set, and the same variable name appears repeatedly in the graph at different nodes or edges, the variable binding is assumed constant. When at most one edge of a given type can occur at each vertex, the labelling can be restricted to only vertices. This is the case for the polynucleotide graphs below. A variable name on an edge then implies that a graph with or without the edge can be matched.

The three types of edges in nucleic acid graphs correspond to the 5' and 3' backbone covalent bonds and the hydrogen bonding mediated base pairing. Both vertex and edge labels will later be associated with a binary value, determining matching behaviour in a simplification of the variable graph concept (simple variable graphs) see section 4 below. For the moment, we introduce the molecular graphs used in the applications without this extension to variables.

Definition 2: A *nucleic acid molecular graph* $G = (V, E, l, \Sigma_{VE})$ is a vertex and edge labelled graph of maximal degree 3, a label set $\Sigma_{VE} = (lab \times type \times B \times rev \times enz) \cup \{prev, next, hybrid\}$ where lab is a set of labels, $type = \{RNA, DNA\}$, $B = \{A, G, C, T/U\}$, $rev = \{0, 1\}$, enz is a set of enzyme labels, and *prev*, *next* and *hybrid* are edge labels. The directed edges of the *next* (*prev*) class point in the 5'-3' (3'-5') direction. The edges labeled *hybrid* join hydrogen bonding bases (base pairs).

For practical graph calculus with nucleic acids it is often convenient to lump together chains of successive identically bonded monomers into oligonucleotides.

Definition 3: A *polynucleotide molecular graph* $G = (V, E, L, \Sigma_{VE})$ is a nucleic acid molecular graph with the *base* attribute of the label set Σ_{VE} replaced by $seq = B^+$, i.e. the set of all nucleic acid base sequences.

While in a *nucleic acid molecular graph* each vertex represents a nucleotide, a vertex in a *polynucleotide molecular graph* represents a sequence of nucleotides. Reverse complementary oligonucleotides can form double strands via hybridisation. With the complement mapping $\bar{\cdot}: B \rightarrow B$ defined by $\bar{A} = T/U, \bar{C} = G, \bar{G} = C, \bar{T} = U$, we can define the reverse complement mapping $B^+ \rightarrow B^+$ as $revcomp(a_1 a_2 \dots a_n) = \bar{a}_n \bar{a}_{n-1} \dots \bar{a}_1$. A polynucleotide molecular graph G with its directed edges is shown in fig. 1. Since between two nodes only these directed „double edges“ exist, from now on we shall display them by simple edges.

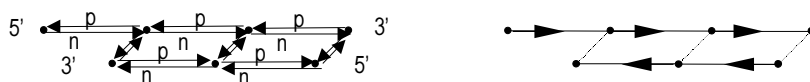


Fig. 1 Polynucleotide molecular graph G : graphical abbreviation. On the left G is with dual directed edges and on the right with the short graphical representation used hereafter. The *prev* (p) and *next* (n) edge pairs are converted to an arrow which points into the 5' to 3' direction. In the data structures however, both these and the *hybrid* edges (dotted) are really two separate edges with opposing directions.

3. Graph Replacement Systems

Chemical reactions changing the structure of molecules can be described on the graph level as deletion/insertion of edges, deletion/insertion of nodes or as a change in attributes. Rules for such reactions can be captured in a graph replacement system. In this section we outline the basic definitions. These are needed for the formulation of reactions on variable molecular graphs in section 4.

A *graph replacement system* consists of one or a set of *starting graphs* and a set of *replacement rules*. A *graph grammar* [6] is a special graph replacement system and generally describes a set of graphs, called its *language*. A graph belongs to the language when it can be *derived* from the starting graph of the graph grammar using the replacement rules. A *derivation* consists of a succession of applications of such rules. Graph replacement systems now play a special role within the general context of *term rewriting systems* in symbolic algebra and the theory of programming languages [3].

In the study of graph grammars it is mainly important to produce the graphs of the language by finite application of rules, or to decide whether a graph belongs to the language or not, whereas in the case of the more general graph replacement systems the precise description of the dynamics of graph changes is significant.

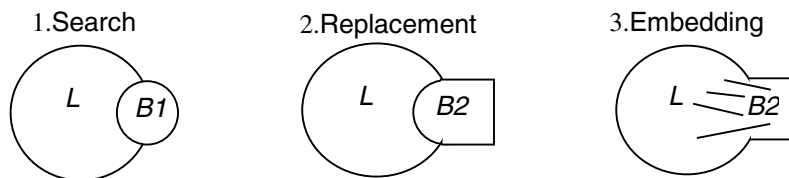


Fig. 2. Application of a rule of a graph replacement system in three steps.

The application of a replacement rule $B1 \Rightarrow B2$ is carried out in three steps:

- search for and eliminate the part of the graph G defined by $B1$, leaving graph L .
- replace the eliminated part by a second graph, defined by the r.h.s. of the rule.
- after the exchange, *embed* the added graph into the rest graph.

The way in which the embedding takes place depends on the *graph replacement model* used. For an algebraic approach see [7]. We make some conditions on our replacement rules $R \Rightarrow P$ (acting on molecular graph M):

- Corresponding vertices in R and P get the same labels. In this way a bijective mapping (identity) is maintained between nodes represented in R and P .
- If a vertex v is deleted in M (resp. built in M) with a replacement rule, we demand that all neighbours $N(v)$ are in R (resp. P), i.e. $V(R) - \{\text{vertices to delete}\} = V(P) - \{\text{new vertices}\}$.

The embedding mechanism of our approach is:

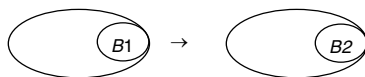
- 1) Match R in M as a subgraph, so we get corresponding vertices of R in M .
[i.e. a mapping $m: V(R) \rightarrow V(M)$ is given].
- 2) For all edges in R , delete the corresponding ones in M .
[$\forall (v,w) \in E(R)$: delete $(m(v), m(w)) \in E(M)$].
- 3) For all vertices v , which are in P but not in R , insert v in M .
[$\forall v \in V(P) \setminus V(R)$: $V(M) = V(M) \cup \{v\}$].
- 4) For all edges in P , insert corresponding edges in M .
[$\forall (v,w) \in E(R)$: $V(M) = V(M) \cup \{(m(v), m(w))\}$].

The theory of graph replacement systems with variable graphs appears somewhat different from that developed within the context of graph or term rewriting systems [2] [3]. In any case, the questions relevant to understanding catalytic DNA reaction networks are not primarily equivalence or reduction to normal form: in general sets of interacting graphs must be considered with the replacement rules themselves being generated by graph operations. This is a further extension beyond programmed structure replacement systems [8].

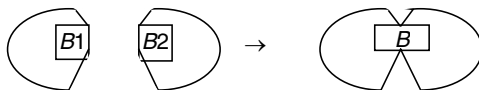
4. Molecular Graph Reactions

We can distinguish the following classes of chemical reactions. Higher order reactions are usually decomposable as a series of at most bimolecular steps.

- *unimolecular reactions*: here only one reactant is involved. The molecule reacts at a local site described by a subgraph $B1$:



- *bimolecular reactions*: here two local sites $B1$ and $B2$ in the molecules react together. New edges



will be created between vertices of subgraphs $B1$ and $B2$:

- *implicit enzymatic reactions*: in the case of enzyme catalysis, where the enzyme is unchanged in the course of the reaction, it is convenient in many cases to treat the reaction like a unimolecular reaction (i.e. ignoring the specific graph structure of the enzyme) and use a vertex tag rather than a new type of labelled edge to indicate the point of attachment of the enzyme. From the point

of view of a coherent theory however it is more natural to treat an enzyme with unknown or constant structure as a single vertex.

The repeated action of local reactions can give rise to complex processes such as polymerisation. Because our description is not restricted to the atomic level, molecular reactions are not simply changes of the molecular structure but also changes of attributes. We build on the graph replacement formalism outlined in the previous section, but because attributes can be processed (e.g. variable binding) the replacement operation must be a little more sophisticated in the case of variable molecular graphs.

For the purposes of defining some classes of molecules, it suffices to restrict the variable graph definition introduced above. In the case of the graph R-OH for example, the R label is a don't care, but in R-O-R it is a true variable implying that the two groups forming the ether must be the same. When this common binding property of variables is not required, it suffices to use a single bit for any attribute to indicate don't care.

Definition 4: A *simple variable graph* is a vertex and edge labelled graph with a label set $\Sigma_{VE} = A_1 \times \dots \times A_n \times \{0, 1\}^n$, where the A_i are a finite number n of sets of attribute values. The i th attribute is attached to the i th binary flag, m_i .

This structure induces a mapping $m: A \rightarrow \{0, 1\}$ which is used to specify constant and don't care attributes for matching. In what follows we use m_v , A_v etc. to denote the above quantities restricted to the vertex v . As an example of such a *simple variable graph*, for a polynucleotide molecular graph we have an induced mapping *match*: $V \rightarrow \{0, 1\}^8$ specifying for each property at each vertex v either *match* or *don't care*:

$$m \in \{0, 1\}^8, m = \begin{array}{cccccccc} m_1 & m_2 & m_3 & m_4 & m_5 & m_6 & m_7 & m_8 \\ \square & \square & \square & \square & \square & \square & \square & \square \end{array} \quad (\text{bit})$$

$$\begin{array}{cccccccc} \text{enz} & \text{rev} & \text{typ} & \text{seq} & \text{lab} & \text{hybr.} & \text{next} & \text{prev} \end{array} \quad (\text{attribute})$$

transferring edge attributes to their adjoining nodes as discussed in section 1. As an example, if a polynucleotide has binary flag 1 at two different nodes, it means that at both these nodes either DNA or RNA oligonucleotides are allowed, independently. This is different from the case of the same variable appearing for this attribute at both nodes, which would instead imply both nodes would have to be an oligonucleotide of the same type (e.g. either DNA or RNA).

Definition 5: A simple variable graph $G = (V, E, \Sigma_{VE})$ *matches* a graph H , if

- $G \subseteq H$ (i.e. $\exists \varphi: V(G) \rightarrow V(H)$) and
- $\forall v \in V(G), m_i(v) = 0 \Rightarrow \alpha_i(v) = \alpha_i(\varphi(v))$.

For every attribute with $m_i(v) = 0$ we have to ensure that its value in G corresponds to the value of the attribute in the matched vertex in H . Moreover there must be a correspondence for those edges with a type attribute $m_t(v) = 0$ between vertices of the

matched subgraph and vertices of the rest graph. In the case of polynucleotide molecular graphs, if at $v \in G$ $m_i(v) = 0$ ($6 \leq i \leq 8$), it is not permitted that the corresponding edge in graph H exists. If $m_i(v) = 1$, the existence of the edge in the graph H is not considered in the matching.

Examples of unimolecular and bimolecular reactions defined by variable graph replacement rules are shown in fig. 3. As an example of the specification of enzymatic reactions using vertex tags we have chosen the RNA transcription carried out for example by the T7 RNA polymerase, see fig. 4. This enzyme along with two others will be used in our application to coupled amplification in section 6.

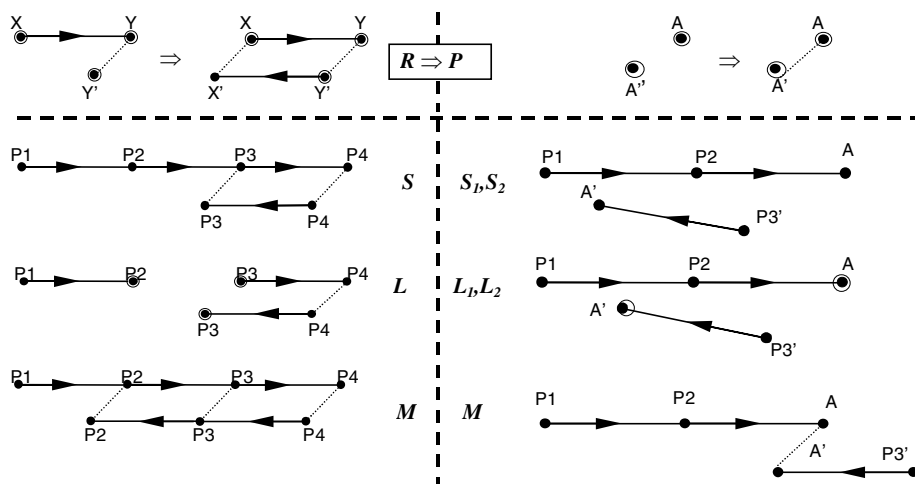


Fig. 3. Unimolecular and Bimolecular Reactions as Subgraph Graph Replacements.

(Left) A unimolecular reaction: template directed polymerisation. The reaction described in the top line involves the local virtual reactant and virtual product, with variable labels marked by circles on the vertices. The reactant S is matched in the centre, edges removed in L and replaced by the virtual product (with bound variables) giving the product M .

(Right) A bimolecular reaction: template hybridisation. The local reaction involves two separate subgraphs. Demanding matching to separate reactants yields a bimolecular reaction.

It is possible to logically distinguish unimolecular from bimolecular reactions and control the order in which reactions occur if desired. For example demanding that all possible unimolecular reactions are completed before further bimolecular reactions often provides a useful chemically founded simplification of the product spectrum, suppressing reactions from short-lived intermediates in unimolecular reaction chains.

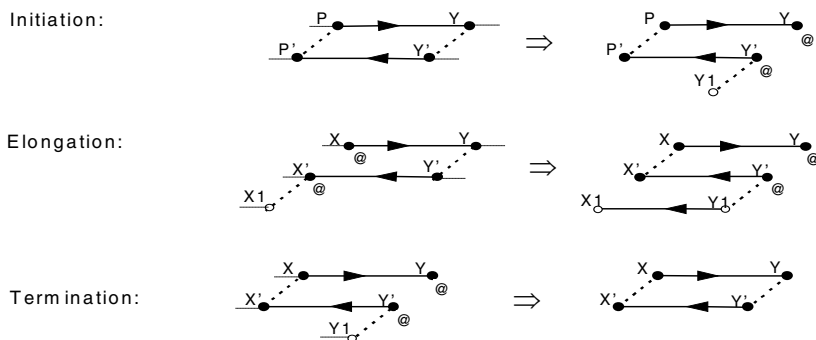


Fig. 4 T7 RNA polymerase action is captured by 3 replacement rules (reactions). Symbols used are: \circ vertex of RNA strand, \bullet vertex of DNA strand, X, Y, Z variable vertex names, @ catalyst binding flag, P T7-RNA-polymerase promoter with the nucleic acid sequence "TAATACGACTCACTATA". An - - - edge in the virtual reactant means a variable bond, i.e. if absent, the matching proceeds iff there is no bond in the molecule. The reaction is initiated by enzyme binding, the binding site(s) being tagged. In the elongation reaction, the enzyme with its marking move across the molecule. Termination involves enzyme separation and tag removal.

5. Molgraph: The Program and IO

We have implemented a program in C for molecular graph reactions. This program reads start molecules in the form of molecular graphs and possible types of reactions in the form of variable graph replacement rules from an ASCII-file. The program computes all combinations of derivations from the start molecules and gives step by step the results of each direct derivation to the screen. While the code is in principle general, in the first instance we have focused our work on the application to nucleic acid processing. The data structure of a molecular graph is captured primarily by the bond-pointers in a vertex structure:

```
struct vertex { struct vertex *bond[NBONDS]; // edges (NBONDS=3 for nucleic acid)
               struct label *lab;           // vertex label
               unsigned match;              // bit mask for vertex fields: variable or instance
               char *complex;               // catalyst binding flag
               struct vertex *rc;           // pointer for reaction complex
               int trace;                   // flag, if vertex was visited
```

where for nucleic acids the label structure is:

```
struct label { char *name;                  // vertex name
               char *sequence               // nucleotide sequence
               int type;                    // vertex type (1:RNA, 2:DNA)
               int reverse;                 // reverse flag
```

For matching purposes, the strands of a molecule (along the covalent backbone of 5'-3' bonds) are stored in an extra strand list. The strands of a molecule are ordered by length and lexicographic order of the vertex label sequence in 5'-3'-direction. The molecules are stored in a molecule list, where a strand pointer points to the first strand of the molecule. The data structure of a replacement rule consist of one (unimolecular

reaction) or two (bimolecular reaction) virtual reaction graphs R_1 and R_2 and a virtual product graph P . The program builds a reactant-product complex PR as in fig. 5.

For the first step (matching) of a direct derivation, we go recursively through the vertex structure of the molecule M and look for a matching. If a subgraph is found, we will set

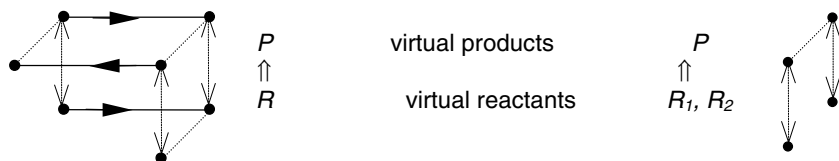


Fig. 6. Two reactant-product complexes. Left a complex made of one virtual reactant and a virtual product (unimolecular reaction). To the right a complex made of two virtual reactants (their graphs are only one vertex) and a virtual product (bimolecular reaction). Pointers between corresponding vertices (same label) of reactant and product build the complex.

pointers from the vertices of the virtual reactant R to the corresponding vertices of M (figure 6). For the second and third step (remove and add), we go recursively through the R and remove all edges in M corresponding to edges in R . Then we proceed recursively through P . Finally, we add the corresponding edges, add vertices if they

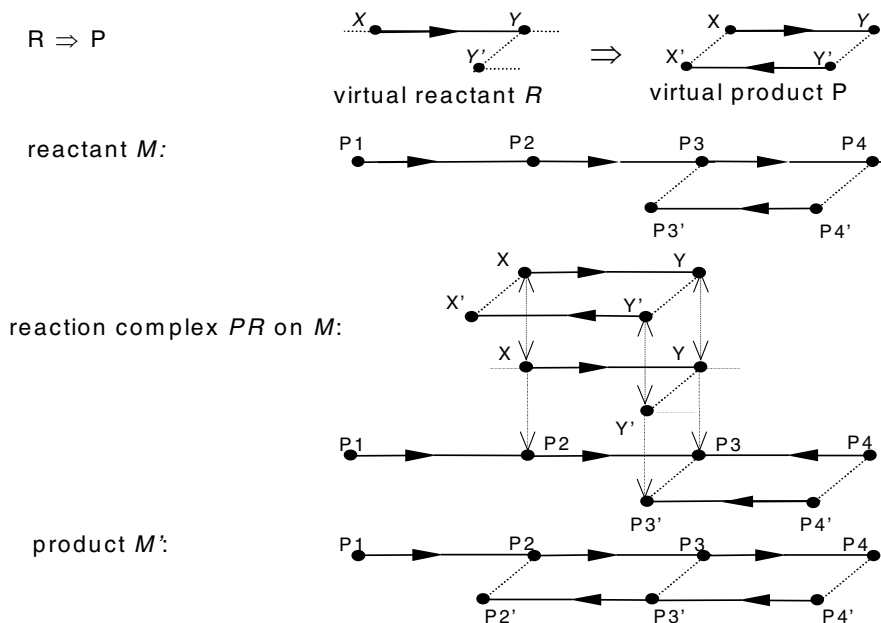


Fig. 5. Computation of a reaction given by the production $R \Rightarrow P$. PR is the reaction complex on molecule M . Note that P cannot match on vertices $P3$, $P4$, $P4'$, because $P4'$ has a NEXT-edge to $P3'$, but in R $match_0(X') = 0$ and X' has no NEXT-edge.

are new in P and change attributes of vertices in M , if required by the match mask of the corresponding vertex in P .

As seen in definition 4, a simple variable graph has at each vertex a match-mask, 8 bits in the case of polynucleotides. Each bit represents one of the vertex attributes (enzyme, reverse, type, sequence, label) and the three possible bonds (HYBRID, NEXT, PREV). The values of a match control bit mean the following:

- in the virtual reaction graph, a bit describes which way the search is to be done. If the bit for an attribute A is set to 0, the attribute values of the corresponding vertices in virtual reaction graph and molecular graph have to be the same. If the bit is set to 1, the attribute value will be ignored in matching. If the bit for an edge is 0, the corresponding vertices of the virtual reaction graph and molecular graph must both either be joined by an edge or not. If the bit is 1, the existence of this edge in the molecule plays no role in the matching. Presently this holds only for edges between the matched subgraph and the rest graph of the molecule.
- in the virtual product graph, a bit describes in which way the attribute of a vertex will be changed. If the bit is 0, the value of an attribute will be determined by its value at the corresponding vertex of the molecular graph. If the bit is 1, the attribute will be set to the value at the virtual product graph vertex. The last three bits for bonds have no meaning here.

The result of a direct derivation will be shown on the screen. The program checks for the occurrence of the product molecules in the molecule list. If a product exists already, a message is given, otherwise it is stored in the list receiving an ID number. The program also recognises whether the computed molecule is one of the start molecules. The program ends either when all reactions are done on all molecules, i.e. there can not be created a new molecule from reactions on the existing molecules, or after a certain number of derivation cycles.

6. Coupled Isothermal Amplification

In this section we present the results of an application of the molecular graph reaction formalism to an experimental system of reactions which has been constructed in our laboratory. Isothermal amplification of polynucleotides can be achieved by a multi-step process called the 3SR reaction [9] involving the joint action of three enzymes: primed reverse transcription of ss-RNA to a DNA-RNA hybrid (RT), digestion of the hybrid RNA (RNase H), polymerisation to ds-DNA (RT) and then repeated transcription of ss-RNA from this DNA (T7 RNA polymerase). The latter enzyme reaction was presented as an example in section 4, fig 5. The other two enzymes are shown in fig. 7.

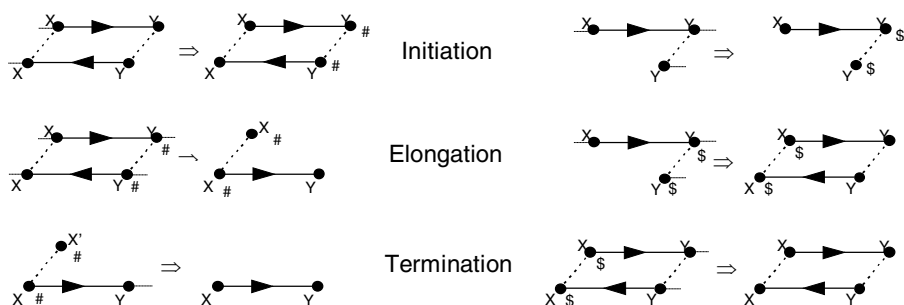


Fig. 7. RNAse H (left) and reverse transcriptase RT (right) as graph replacement rules.

An example of the textual input for the T7 elongation reaction is shown below:

```
@T7-Elongation
# 1st virtual reactant
X 2 1 01111101 0000000 $
Y 2 1 01111010 0000000
Y' 2 2 01111001 0000000
X' 2 1 01111010 0000000 $
X1 1 2 01011001 0000000
# 2nd virtual reactant
NULL
```

	#	virtual	product
Y	2 1	01111111	0000000 \$
X	2 0	01111111	0000000
X'	2 2	01111111	0000000
Y'	2 0	01111111	0000000 \$
Y1	1 2	00110111	0000000
X1	1 0	01111111	0000000

Format: label [sequence] type nbond match compare [enzyme symbol]
 type [1=RNA, 2=DNA]
 nbond [0=5', 1=3', 2=Hybrid]
 match, compare (mask: enz, rev, type, seq, lab, hybrid, next, prev)
 for match: [1=don't care, 0=match], for compare: [1=neq, 0=eq]

This reaction scheme has been used as the basis for deriving experimental coupled amplification schemes capable of novel evolutionary behaviour. As a practical example of the utility of the present approach we have calculated the complete spectrum of reactions starting with the templates and primers for the CATCH system [10] involving the bi-molecular step of template-template hybridisation. The molecules produced from the iterated application of the 3 enzyme reactions and hybridisation are shown in fig. 8.

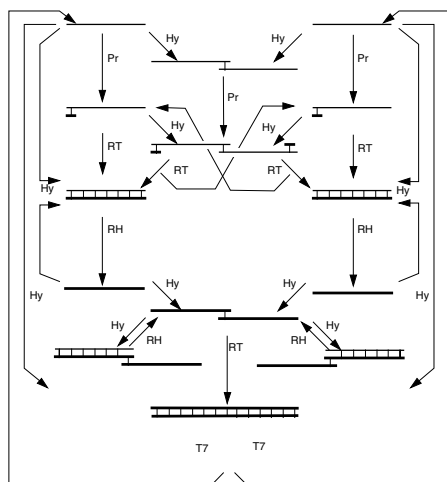


Fig. 8. Coupled amplification of the CATCH scheme with side reactions found by the program MOLGRAPH.

This scheme is the result of an application of the above molecular graph replacement chemistry to the CATCH reaction.

Hy = hybridisation, Pr = primer hybridisation, RT = reverse transcriptase, RH = RNAse H, thin lines RNA, thick lines DNA. The thin vertical lines represent base-pairing bonds. The several vertices forming chains of oligonucleotides in the single stranded DNA and RNA are not shown explicitly. The RNA starting species for example were P_1' -Pr₁-Sp₁-Hy and P_2' -Pr₂-Sp₂-Hy'.

7. Outlook

We have demonstrated two novel capabilities in this work. Firstly, a formalisation of the structural chemistry of DNA processing (and other informational macromolecules) can be constructed, allowing the investigation of combinatorial networks of catalysed reactions in an automated fashion. A suitable level of abstraction has been employed to this end. Secondly, biochemists can specify known enzymatic reactions succinctly in terms of their local processing effects and have now a tool for the investigation of the consequences of the concerted action of DNA/RNA processing enzymes. This may be important for the entire field of molecular biology, but especially for an exploitation of its applications to DNA computing [11].

At present the program computes all possible derivation paths as long as new molecules are formed. Some DNA computations will also require a more synchronously clocked treatment for a finite number of iterations, which is readily implemented. The recognition of catalytic cycles must consider not only the production of products previously used as reactants in the course of a reaction path (or derivation), but also the consumption of these reactants. A simple interconversion cycle such as $A \rightarrow B \rightarrow A$ is not replication. Such analysis will require the storage of entire derivation trees. The analysis of the reaction graphs created by the molecular graph chemistry constructed in this work must form the subject of a separate investigation. The logical identification of self-amplification cycles for example is one application.

The application to DNA Computing and molecular self-organisation involves a search for minimal basis sets of local reactions which can both assemble complex sets of molecular graphs from simple starting molecules, and organise the catalytic cycles to concentrate the products in particular families. A complete analysis of this requires both structural and kinetic considerations. Since the specification of the local reactions is quite simple, however, we expect a combinatorial search of the molecular

graph families produced by simple sets of local reaction rules to be informative and within the reach of the tools developed here. This will be the subject of future work.

In contrast with more formal approaches such as [4], the present work allows a direct relation between abstract structural computations and real chemistry. The example of the 3SR reaction illustrates both the problems and potential of DNA computations involving structural complexes of DNA: the rich array of side reactions for enzymatically defined processes can lead to combinatorially complex libraries even for comparatively simple reactions. Although at present the theory is applied to combinatorial reactions on nucleic acids, it can be readily extended to other classes of informational molecules. The use of local variable graph replacement rules to allow the user to specify reaction classes succinctly and then explore the product space may find general application in DNA processor design and error analysis. The next more detailed level would involve reaction prediction programs in organic synthesis [2], which is currently not tractable for DNA, but there are some grounds for optimism that a wide range of combinatorial reaction families can be handled iteratively at this level.

Acknowledgements: The financial support of the German Ministry (BMBF Gr. #0310799) during an early phase of this work is gratefully acknowledged. The authors wish to thank R. Ehricht and T. Ellinger for discussions concerning the application to the 3SR reaction and U. Tangen, J. Ackermann and T. Rucker for correcting the final manuscript.

References

- [1] Thue, A.: Probleme über Veränderungen von Zeichenreihen nach gegebenen Regeln, Skr. Vid. Kristianaia, I. Mat. Naturv. Klasse **10** (1914) 34pp.
- [2] Rozenberg, G. Ed. : Handbook of graph grammars and computing by graph transformation. (World Scientific, Sing.-NJ-London-Hong Kong, 1997).
- [3] Narendran, P. and Rusinowitch, M. Ed.: Rewriting Techniques and Applications. LNCS **1631** (Springer, Berlin 1999).
- [4] Fontana, W.: Algorithmic Chemistry. Technical Report LA-UR 90-1959, Los Alamos Natl. Lab. (1990).
- [5] König, Denes: Theorie der endlichen und unendlichen Graphen (Teubner, Leipzig 1986).
- [6] Nagl, Manfred: Bibliography on Graph Rewriting Systems (Graph Grammars, 1983).
- [7] Ehrig, Harmut: Tutorial introduction to the algebraic approach of graph grammars. - LNCS **291** 3-14 (Springer, Berlin 1987).
- [8] Schürr, A.: Programmed graph replacement systems, pp479-546 in Rozenberg, G. Ed. Handbook of graph grammars and computing by graph transformation. (World Scientific, Sing-NJ-London-Hong Kong, 1997).

-
- [9] Fahy, E., Kwoh, D.Y. and Gingeras, T.R: Self-sustained sequence replication (3SR): an isothermal transcription-based amplification system alternative to PCR. In "PCR Methods and Applications" pp25-33 (Cold Spring Harbor Lab. Press: NY, 1991).
 - [10] Ehricht, R., Ellinger T., and McCaskill, J.S.: Co-operative amplification of templates by cross hybridisation (CATCH). *Europ. J. Biochem.* **243** 358-364 (1997).
 - [11] Adleman, L.M.: Molecular computation of solutions to combinatorial problems. *Science* **266** 1021-1023 (1994).
 - [12] Ihlenfeldt, W. and Gasteiger, J.: Computer-assisted planning of organic synthesis: the second generation of programs. *Angew. Chem. Int. Ed. Engl.* **34** 2613-2633 (1995).

DNA and Circular Splicing^{*}

Paola Bonizzoni¹, Clelia De Felice², Giancarlo Mauri¹, and Rosalba Zizza^{1**}

¹ Dipartimento di Informatica Sistemistica e Comunicazione
Università degli Studi di Milano - Bicocca
Via Bicocca degli Arcimboldi 8, 20126 Milano - Italy
{bonizzoni, mauri}@disco.unimib.it
rosziz@bioinformatics.bio.disco.unimib.it

² Dipartimento di Informatica ed Applicazioni,
Università di Salerno, 84081 Baronissi (SA), Italy
defelice@unisa.it

Abstract. Circular splicing has been very recently introduced to model a specific recombinant behaviour of circular DNA, carrying on the investigation initiated with linear splicing. In this paper we restrict ourselves to the relationship between circular regular languages and circular splicing languages. We provide partial results towards a characterization of the class of circular regular languages generated by finite circular splicing systems. We consider a class of languages X^* closed under conjugacy relation and with X a regular languages, called here *star languages*. Using automata theory and combinatorial techniques on words, we show that for a subclass of star languages the corresponding circular languages are circular (Paun) splicing languages. In particular, star languages with X being a finite set or X^* being a free monoid belong to this subclass.

1 Introduction

The notion of *splicing systems* was firstly introduced in [11], where T. Head modelled a recombinant behaviour of DNA molecules (under the action of restriction and ligase enzymes) as a particular operation between words in a formal language and so suggesting the possibility of using molecules to perform computations. Since then, the way from DNA Recombination to DNA Computing via formal languages has been largely explored for both theoretical and technological implications. In this context, the *splicing* operation has been used to model specific recombinant behaviours. In short, a restriction enzyme is able to recognize a pattern in a DNA molecule and to cut this molecule inside the pattern in a specific position, so providing two segments of DNA. Then a ligase enzyme binds together pairs of sequences coming from different molecules, so generating a new molecule. In order to treat this cut and paste phenomenon in

^{*} Partially supported by MURST Project “Unconventional Computational Models: Syntactic and Combinatorial Methods” - “Modelli di calcolo innovativi: Metodi sintattici e combinatori”.

^{**} E-mail address for correspondence: `rosziz@bioinformatics.bio.disco.unimib.it`

formal language theory, an initial set of strings (the initial set of DNA molecules) and a set of splicing rules or special words (simulators of enzymes behaviour) are given, such that the set of all possible molecules generated by the biochemical process of cut and paste is nothing but the language of all possible words that may be generated by applying the rules to the strings. We do not take into account the bidimensional nature of DNA even if very recently a link has been given between splicing systems and picture languages (see [10] for a survey on picture languages). We also deliberately ignore other molecular considerations and basic details (for a complete monograph see [17]).

DNA occurs in both linear and circular form and, correspondingly, linear and circular splicing systems have been defined. The introduction of circular splicing rules may be justified as follows: an example of this type of splicing occurs in a recombinant mechanism (transposition) between bacteria and plasmides. Moreover, the replication is possible only if DNA has a circular form in the host cell. Furthermore, it could be interesting to use circular DNA in Adleman's experiment, since plasmides autoreplicate themselves without errors and exponential weight increasing, which are weak points in Adleman's approach [1]. In this paper we mainly deal with circular splicing systems, providing interesting links with the linear case, which has been deeply investigated. Furthermore, in our theoretical model there is no hypothesis on the number of copies of the initial set of molecules.

Classical results in formal language theory and combinatorial tools have been of great use for reaching the results given in this paper. It is well known that three definitions of (linear) splicing systems are given by Head, Paun and Pixton. The computational power of these systems, i.e. the class of languages which are generated by them, has been completely described. This computational power depends on which level in the Chomsky hierarchy the initial set I and the set of the rules R belong to. In particular, if the last two sets are finite, the splicing-generated languages are a subclass of the regular languages, completely characterized in [4]. We now state some results towards the proof of a counterpart of this characterization for circular splicing. Precisely, three corresponding definitions of circular splicing operations have been given (by Head, Paun and Pixton). We show that under an additional hypothesis, for a star language (i.e. a language X^* closed under conjugacy relation and with X a regular language), the corresponding circular language is generated by a circular Paun splicing system with a finite initial set I and a finite set R of rules (Theorem [1]). Furthermore, this result finds a deeper insight in the framework of the theory of variable-length codes [2]. Indeed, we prove that for all star languages X^* with X being a finite set or X being a rational code, the corresponding circular languages are circular splicing languages (Proposition [5] and Corollary [1]). Observe that in contrast with the linear case, a finite initial set and a finite set of rules do not guarantee that the splicing-generated language is regular.

Let us briefly sketch the contents of this paper. In Section [2] we gathered an overview on linear splicing and easy new and known results on circular splicing. In the same section, we also explicitly observe that the computational power of

the three circular splicing systems is already known to be different. As far as we know, very few results on circular splicing systems are known, and these are surveyed in Sections 2 4. For the sake of completeness, in the same Section 4 we reported several additional hypotheses which could or could not be added to circular splicing systems. Some questions are also asked. In our results, presented in Section 3, we restrict ourselves to Paun's definition with no additional hypotheses. All the proofs which are not contained in this paper may be found in an extended version of the paper 5.

2 Languages and Splicing: Definitions

In the next part of this paper, we denote by A^* the free monoid over a finite alphabet A . We also set $A^+ = A^* \setminus 1$, where 1 is the empty word.

2.1 Linear Splicing

In this paper we only deal with circular splicing, but for completeness, let us briefly sketch some concepts and results on linear splicing. Literature presents three definitions of the linear splicing operation [11,16,18]. One of them has been introduced by Paun, where the biological process presented in Section 1 has been formalized as follows [16]. The enzymes are encoded by a *splicing rule* $r = u_1|u_2\$u_3|u_4$, for $u_i \in A^*$ and $|\$, \notin A$; the molecules are sequences $w', w'' \in A^*$. If $w' = x_1u_1u_2x_2$ and $w'' = y_1u_3u_4y_2$, the rule r can be applied to w' and w'' , producing two new sequences $w = x_1u_1u_4y_2$ and $z = y_1u_3u_2x_2$. A splicing system S consists of a set R of rules (enzymes) and an initial language $I \subseteq A^*$ (molecules). The global recombinant behaviour of DNA molecules is described by the splicing language $L(S)$ generated by S . $L(S)$ is the smallest set of strings s.t. $I \subseteq L(S)$ and if w', w'' are in $L(S)$ and r is a rule that can be applied to w' and w'' , producing w, z , then w and z must also be in $L(S)$. Let us denote $H(F_1, F_2)$ the class of languages generated by splicing systems $S = (I, R)$, where $I \in F_1$, $R \in F_2$ and F_1, F_2 are classes of languages. It is known that splicing systems can reach the same power of Turing Machines [16, 12]. As we have already said, when we restrict ourselves to splicing systems with a finite set R of rules and a finite set I of strings, we get a class $H(Fin, Fin)$ which is a proper subclass of regular languages, as shown in [8,9,14,18]. Despite the interesting results obtained in this framework, no algorithm was yet known for deciding whether a regular language is generated by splicing. In [4] the first decision procedure to test whether a regular language can be generated by the splicing operation is given, deriving a complementary procedure to the ones proposed in [8,9,14,18]. A very interesting consequence of this result is that this procedure allows us to compare the generative power of different splicing systems. Indeed, in [7] the authors point out that there is a regular language generated by Paun's (resp. Pixton's) splicing operation that cannot be generated by Head's (resp. Paun's) splicing operation.

2.2 Circular Languages

Circular words have already been examined in formal language theory. Indeed, for a given word $w \in A^*$, a circular word $\sim w$ is the equivalence class of w w.r.t. the *conjugacy* relation \sim defined by $xy \sim yx$, for $x, y \in A^*$ (see [15]). A^\sim is the set of all such circular words over A , i.e. the quotient of A^* w.r.t. \sim . Given $L \subseteq A^*$, $Cir(L) = \{\sim w | w \in L\}$ is the *circularization* of L , i.e. the set of all circular words corresponding to elements of L , while any language L such that $Cir(L) = C$, for a given circular language $C \subseteq A^\sim$, is called a *linearization* of C . The set of all strings in A^* corresponding to elements of C is the *full linearization* of C , written $Lin(C)$. Circular splicing deals with circular strings and circular languages and as a result with formal languages which are full linearizations of circular languages. This latter representation will be used in the next part of this paper, where we will consider languages closed under conjugacy and the action of circular splicing over their circularization. We denote Fin (resp. Reg, CF) the class of finite (resp. regular, context free) languages. Given a family of languages FA in Chomsky hierarchy, FA^\sim consists of all those circular languages C which have some linearization in FA . In particular, $C \in Reg^\sim$ if and only if its full linearization is regular [13,18,20]. In this paper we mainly deal with Reg^\sim .

2.3 Circular Splicing

The splicing operation in the circular case deals with biological phenomena which are different from the ones modelled in the linear case, but the three definitions given for linear splicing systems can be adapted to the circular case.

Head's definition [20]. A *circular (Head) splicing system* is a 4-tuple $SC_H = (A, I, T, P)$, where $I \subseteq A^\sim$ is the initial circular language, $T \subseteq A^* \times A^* \times A^*$ and P is a binary relation on T , such that, if $p, x, q, u, y, v \in A^*$, $(p, x, q), (u, y, v) \in T$ and $(p, x, q)P(u, y, v)$ then $x = y$. Thus, given $\sim hpxq, \sim kuxv \in A^\sim$ with $(p, x, q)P(u, x, v)$, the splicing produces $\sim hpxvkuxq$.

Paun's definition [16]. A *circular (Paun) splicing system* is a 3-tuple $SC_{PA} = (A, I, R)$, where $I \subseteq A^\sim$ is the initial circular language, $R \subseteq A^*|A^* \$A^*|A^*$, with $|, \$ \notin A$, is the set of rules. Then, given a rule $r = u_1|u_2\$u_3|u_4$ and two circular words $\sim hu_1u_2, \sim ku_3u_4$, the rule cuts and linearizes the two strings obtaining u_2hu_1 and u_4ku_3 , and pastes and circularizes them obtaining $\sim u_2hu_1u_4ku_3$.

Pixton's definition [18]. A *circular (Pixton) splicing system* is a 3-tuple $SC_{PI} = (A, I, R)$ where A is a finite alphabet, $I \subseteq A^\sim$ is the initial circular language, $R \subseteq A^* \times A^* \times A^*$ is the set of rules. R is s.t. if $r = (\alpha, \alpha'; \beta) \in R$ then $\bar{r} = (\alpha', \alpha; \beta') \in R$. Thus, given two circular words $\sim \alpha h, \sim \alpha' k$, the two rules r, \bar{r} cut and linearize the two strings, obtaining $h\alpha, k\alpha'$, and then paste, substitute and circularize them, producing $\sim h\beta k\beta'$.

Remark 1. We must note that in the original definition of circular splicing language given by Paun in [12], rules in R can be used in two different ways: one way has been described above, the other, called *self-splicing*, will be defined in

Section 4. In this paper in order to make the three definitions uniform, we have preferred to delete self-splicing from Paun's definition.

We now give the definition of circular splicing languages. For a given splicing system SC_X , with $X \in \{H, PA, PI\}$, we denote $(w', w'') \vdash_r w$ the fact that w is produced from (or spliced by) w', w'' by using a rule r . Given a language $C \subseteq A^*$, we denote $\sigma(C) = \{z \in A^* \mid (w', w'') \vdash_r z, \text{ for } w', w'' \in C \text{ and } r \in R\}$. Thus, we define $\sigma^0(C) = C$, $\sigma^{i+1}(C) = \sigma^i(C) \cup \sigma(\sigma^i(C))$, $i \geq 0$, and then $\sigma^*(C) = \bigcup_{i \geq 0} \sigma^i(C)$.

Definition 1. Given a splicing system SC_X , with $X \in \{H, PA, PI\}$, the circular language $C(SC_X) = \sigma^*(I)$ is the language generated by the system SC_X , I being the initial language in SC_X .

A circular language C is CX generated (or C is a circular splicing language) if a splicing system SC_X exists such that $C = C(SC_X)$.

2.4 On the Generating Power of Circular Splicing Systems

In this paper we deal with Paun's systems $SC_{PA} = (A, I, R)$, with both I, R finite sets, and with the corresponding class of generated languages, denoted $C(Fin, Fin)$. As a matter of fact, the investigation about circular splicing systems is usually limited to ones with a finite or regular initial language. Furthermore, it goes without saying that the set of rules is finite in each definition.

Example 1. In [20] it is shown that $(aa)^*$ is not a linear splicing language, while the corresponding circular language $\{\sim(aa)^n \mid n \geq 0\}$ is CH generated by $SC_H = (A, I, T, P)$ with $A = \{a\}$, $I = \{\sim 1, \sim aa\}$, $T = \{(1, a, 1)\}$, and $(1, a, 1)P(1, a, 1)$. We can observe that $\{\sim(aa)^n \mid n \geq 0\}$ is CPA generated, by choosing $A = \{a\}$, $I = \{\sim aa\}$, $R = \{aa|1\$1|aa\}$. On the contrary, let us consider the regular language $(aa)^*b$. Using results in [4], we get $(aa)^*b \in H(Fin, Fin)$. Take its circularization $C = \sim(aa)^*b$. Notice that there is a linearization of C , $\{a^kba^k : k \in \mathbb{N}\}$ which is not regular. However, the full linearization is $(aa)^*\{b, aba\}(aa)^*$, which is regular and given in [18]. Thus $C \in Reg^*$, in virtue of the definition [18, 20]. We immediately see that C cannot be CH or CPA generated (Proposition 2). Nevertheless, it is not too difficult to prove that C is CPI generated, when we choose, for example, $A = \{a, b\}$, $I = \{\sim b, \sim a^2b, \sim a^4b\}$, $R = \{(a^2, a^2b; a^2), (a^2b, a^2; 1)\}$.

The previous example shows the difference between linear and circular splittings. The same example also suggests the problem, formalized below, of comparing the computational power of the three definitions of the circular splicing operation given in Section 2.3.

Problem 1. Given $C = C(SC_X)$, with $X \in \{H, PA, PI\}$, does SC_Y exist, with $Y \neq X$, $Y \in \{H, PA, PI\}$, such that $C = C(SC_Y)$?

It is not too difficult to prove Proposition 1. This proposition and Example 1 show that the computational power of Pixton's systems is greater than the other two. This gives a partial answer to Problem 1.

Proposition 1. *If $C \subseteq A^\sim$ is CH generated, then C is CPA generated. Moreover, if $C \subseteq A^\sim$ is CPA generated, then C is CPI generated.*

Proposition 2, quoted in Example 1, allows us to state that some circular languages are not generated by Head's or Paun's systems.

Proposition 2. *Let $L = \text{Lin}(C)$ be an infinite language which is the full linearization of a circular language C . Suppose that L satisfies the following condition*

$$\forall l, m \in L : \quad lm \notin L.$$

Then $\text{Cir}(L)$ is not CPA generated.

Example 2. By using Proposition 2, we can easily state that some circular regular languages cannot be generated by circular Paun's splicing systems. For instance, let $L_1 = \{w \in A^* \mid \exists h, k \in \mathbb{N} \mid |w|_a = 2k, |w|_b = 2h\}$ and $L_2 = \{w \in A^* \mid \exists h, k \in \mathbb{N} \mid |w|_a = 2k+1, |w|_b = 2h+1\}$. L_1 and L_2 are languages closed under conjugacy. Moreover, L_1 and L_2 are regular languages (L_1 is the shuffle of $(aa)^*$ and $(bb)^*$, L_2 is the shuffle of $(aa)^*a$ and $(bb)^*b$). Thus, $\text{Cir}(L_1)$ and $\text{Cir}(L_2)$ are circular regular languages. By using Proposition 2, we can see that $\text{Cir}(L_2)$ is not CH or CPA generated. On the contrary, we will see in Section 3 that $\text{Cir}(L_1)$ is CPA generated.

3 Regular CPA Generated Languages and Star Languages

It is already known that in contrast with the linear case, $C(\text{Fin}, \text{Fin})$ is not intermediate between two classes of languages (in the Chomsky hierarchy). For example, $\sim a^n b^n$ is a circular context-free language which is not a circular regular language (since it has no regular linearization), but $\sim a^n b^n$ is CH generated with finite initial circular language [20]. On the other hand, we have already seen examples of regular circular languages that cannot be generated by using CPA splicing systems with finite initial circular language (see Example 1). So far, we have not yet discovered whether $C(\text{Fin}, \text{Fin})$ contains any context-sensitive or recursive enumerable language which is not context-free. We believe that $C(\text{Fin}, \text{Fin}) \subseteq NP$. Nevertheless, in this paper we restrict ourselves to the following problem.

Problem 2. Characterize $\text{Reg}^\sim \cap C(\text{Fin}, \text{Fin})$.

We approach Problem 2 by dealing with the class of languages defined below.

Definition 2. *A star language is a language $L \subseteq A^*$ that satisfies the following conditions:*

- (1) $L = X^*$, with X a regular language,
- (2) L is closed under conjugacy relation.

Under an additional hypothesis, a star language has its circularization in $C(\text{Fin}, \text{Fin})$, as shown in Theorem 1, whose proof can be obtained thanks to the following proposition.

Proposition 3. *Let X^* be a star language and let $SC_{PA} = (A, I, R)$ be a splicing system. If $I \subseteq \text{Cir}(X^*)$ then $C(SC_{PA}) \subseteq \text{Cir}(X^*)$.*

Proof. Let us prove that $C(SC_{PA}) \subseteq \text{Cir}(X^*)$ holds, by using induction over the length $|y|$ of $\sim y \in C(SC_{PA})$. If $\sim y \in I$ then $\sim y \in \text{Cir}(X^*)$, since $I \subseteq \text{Cir}(X^*)$. Otherwise, $y' \in \sim y$ exists such that $y' = u_2hu_1u_4ku_3$ with $\sim u_2hu_1, \sim u_4ku_3 \in C(SC_{PA})$ and $\sim u_2hu_1 \neq \sim y, \sim u_4ku_3 \neq \sim y$. Then $|u_2hu_1| < |y|, |u_4ku_3| < |y|$. By induction hypothesis, $\sim u_2hu_1, \sim u_4ku_3 \in \text{Cir}(X^*)$ and, X^* being closed under conjugacy, $u_2hu_1, u_4ku_3 \in X^*$. Thus, X^* being a submonoid, $u_2hu_1u_4ku_3 = y' \in X^*$, and so $\sim y \in \text{Cir}(X^*)$. \square

In Theorem 1, for a star language X^* satisfying an additional hypothesis, we state how $\text{Cir}(X^*)$ is CPA generated (with I, R finite sets). We also use the following notations and we suppose the reader is familiar with elementary finite state automaton theory (see [213]). Let $\mathcal{A} = (A, Q, \delta, i, F)$ be a finite state automaton. \mathcal{A} is *trim* if each state is accessible and coaccessible. A word $c \in A^+$ is a *cycle* (resp. *simple cycle*) in \mathcal{A} if $q \in Q$ exists such that $\delta(q, c) = q$ and the internal states crossed by the transition are different from q (resp. each other and w.r.t. q). Given a regular language L and a finite state automaton \mathcal{A} recognizing L , we inductively define the *fingerprint* $F_{n_c}(c)$ of a cycle c in \mathcal{A} and the class of *fingerprint closed languages* L w.r.t. \mathcal{A} .

Definition 3 (Fingerprint of a cycle). *Let \mathcal{A} be a finite state automaton recognizing a regular language L , i.e. $L = L(\mathcal{A})$. The set $\mathcal{C}(L) = \bigcup_{c \text{ cycle}} F_{n_c}(c)$ of the fingerprints $F_{n_c}(c)$ for any cycle c in \mathcal{A} is defined inductively as follows:*

- if c is simple, then $F_{n_c}(c) = \{c^{n_c}\}$, for minimal $n_c > 0$ s.t. $c^{n_c} \in L$, otherwise $F_{n_c}(c) = \emptyset$;
- if c is not simple, i.e. $c = u_1c_1^{p_1}u_2c_2^{p_2}\dots u_kc_k^{p_k}u_{k+1}$, where u_i are labels of transitions in which no state is crossed twice, $u_1, u_{k+1} \neq 1$, c_i are cycles, $p_i \geq 1$, $F_{n_{c_1}}(c_1), \dots, F_{n_{c_k}}(c_k)$ are nonempty fingerprints of c_1, \dots, c_k , then $F_{n_c}(c) = \{(u_1x_1^{t_1}u_2x_2^{t_2}\dots u_kx_k^{t_k}u_{k+1})^{n_c} \mid x_i \text{ s.t. } x_i^{n_{c_i}} \in F_{n_{c_i}}(c_i), 0 \leq t_i \leq n_{c_i}, i = 1, \dots, k\}$, for minimal $n_c > 0$ such that $F_{n_c}(c) \subseteq L$, otherwise $F_{n_c}(c) = \emptyset$.

Let c be a not simple cycle. Observe that $F_{n_c}(c) = \emptyset$ if either there exists j s.t. $F_{n_{c_j}}(c_j) = \emptyset$, or there exist $x_i^{n_{c_i}} \in F_{n_{c_i}}(c_i)$, $0 \leq t_i \leq n_{c_i}$, $i = 1, \dots, k$ s.t. $(u_1x_1^{t_1}u_2x_2^{t_2}\dots u_kx_k^{t_k}u_{k+1})^* \cap L = \emptyset$.

For a given finite state automaton \mathcal{A} , we say that $L = L(\mathcal{A})$ is a *fingerprint closed language* w.r.t. \mathcal{A} , whenever $F_{n_c}(c) \neq \emptyset$, for any cycle c in \mathcal{A} . Loosely speaking, if L is a fingerprint closed language we are sure that for any label of closed path we have a finite crossing of this label in L .

Thus we can prove our main result.

Theorem 1. *Let X^* be a fingerprint closed star language w.r.t. a trim deterministic automaton \mathcal{A} recognizing X^* . Then $\text{Cir}(X^*) \in C(\text{Fin}, \text{Fin})$.*

Sketch of the proof. Let us consider $SC_{PA} = (A, I, R)$ where I and R are finite sets defined by means of $\mathcal{C}(L)$. Precisely, we define I_1 as the set containing $\mathcal{C}(L)$, along with the labels of the successful paths in \mathcal{A} in which each cycle c is crossed at most n_c times: $I_1 = \{w \in A^* \mid \delta(i, w) \in F \text{ and } \forall x, z \in A^*, c \in A^+, q \in Q \text{ if } w = x\alpha^k z \text{ with } \delta(i, x) = q, \delta(q, c) = q, \alpha \text{ s.t. } \alpha^{n_c} \in F_{n_c}(c), \text{ then } k \leq n_c\} \cup \mathcal{C}(L)$. Let $I = \text{Cir}(I_1)$. Note that I_1 is finite and $I = \text{Cir}(I_1) \subseteq \text{Cir}(X^*)$. Furthermore, we set $R = \{1|1\$1|\alpha; \alpha \in F_{n_c}(c), F_{n_c}(c) \subseteq \mathcal{C}(L), c \text{ cycle}\}$.

Let us denote $C = C(SC_{PA})$ and let us prove that $C = \text{Cir}(X^*)$.

We will firstly show that $\text{Cir}(X^*) \subseteq C$. Let $\sim y \in \text{Cir}(X^*)$. By induction on $|y|$, we prove that $\sim y \in C$, i.e. $\sim y$ is generated by the splicing system given above. By definition, y is the label of a successful path: if $\sim y$ does not contain cycles or each cycle c is crossed at most n_c times (for n_c the one given in the definition of I_1), then, by construction of I , $\sim y \in I$, and so $\sim y \in C$ (by definition). Otherwise, each element $y \in \sim y$ is a successful path in which a cycle c is crossed more than n_c times (for n_c the one given in the definition of I_1). Precisely, $y = xc^r z$ with $x, z \in A^*$, $\delta(i, x) = q$, $\delta(q, c) = q$, $\delta(q, z) \in F$ and $r > n_c$, i.e., $r - n_c > 0$. Choose x with $|x|$ maximal w.r.t. this condition. Since y is a successful path, then $y' = xc^{r-n_c} z$ is a successful path and, by induction hypothesis, $\sim y' \in C$.

We claim that $c^{n_c} \in \mathcal{C}(L)$, where we refer to the occurrence of the word c as the cycle c in the successful path y . This allows us to prove that $\sim y \in \text{Cir}(X^*)$. Indeed, if $c^{n_c} \in \mathcal{C}(L)$, then $\sim c^{n_c} \in I$. By splicing definition, $\sim c^{n_c} \in C$. Consider the linearization zxc^r of $\sim y$. Set $zxc^r = u_2 h u_1 u_4 k u_3$, with $h = zxc^{r-n_c}$, $u_1 = u_2 = k = u_3 = 1$, $u_4 = c^{n_c}$. Clearly, $u_1 | u_2 \$ u_3 | u_4 = 1 | 1 \$ 1 | c^{n_c} \in R$. Moreover, $\sim u_2 h u_1 = \sim zxc^{r-n_c} \in C$ and $\sim u_4 k u_3 = \sim c^{n_c} \in C$: by splicing definition $\sim zxc^r = \sim y \in C$.

We now prove the above stated claim: $c^{n_c} \in \mathcal{C}(L)$. Otherwise, by Definition 3, c is not a simple cycle in y and c^{n_c} is not a fingerprint of the cycle c in y . Then $c = u(c')^t v$, where c' is a cycle in y , $t > n_{c'}$ and $u, v \neq 1$. Thus $y = x(u(c')^t v)^r z$ with $|xu| > |x|$, against the maximality of $|x|$.

Vice versa, $C \subseteq \text{Cir}(X^*)$ follows by using Proposition 3, since $I \subseteq \text{Cir}(X^*)$, i.e. $C = \text{Cir}(X^*)$. \square

Example 3. Star languages exist which do not satisfy the hypothesis contained in Theorem 1. For instance, consider $L = A^* \setminus a^* = (a^* b a^*)^*$ over a two-letter alphabet $A = \{a, b\}$. It is clear that L is a star language. Furthermore, a trim deterministic automaton \mathcal{A} exists such that a is a cycle in \mathcal{A} and $a^* \cap L = \emptyset$. However, L is not CPA generated. Indeed, by contradiction suppose that $C = \text{Cir}(L) = C(SC_{PA})$ with $SC_{PA} = (A, I, R)$. Since $\sim(a^* b) \subseteq C$, $n \in \mathbb{N}$ exists such that $w = \sim(a^n b) \in C \setminus I$. Thus, $\sim h u_1 u_2 \in C$, $\sim k u_3 u_4 \in C$ also exist such that $w = \sim u_2 h u_1 u_4 k u_3$. Since we have only one occurrence of b 's in w , we get $\sim h u_1 u_2 \in \sim(a^*)$ or $\sim k u_3 u_4 \in \sim(a^*)$, a contradiction.

As far as we know, the structure of the regular languages which are closed under the conjugacy relation is unknown, even if we restrict ourselves to the simple case of languages which are the Kleene closure of a regular language. Nevertheless, this structure has been completely described in [3, 19] when X^* is

a free monoid. The necessary definitions for recalling this result can be found in [2]. We briefly report them below.

An algebraic description of some subclasses of the class of the regular languages $L \subseteq A^*$ has been given by means of the *syntactic monoid* $\mathcal{M}(L)$ of L . This is the quotient of A^* w.r.t. the syntactic congruence \equiv_L , defined as follows: $w \equiv_L w'$, with $w, w' \in A^*$, if and only if $xyw \in L \Leftrightarrow xw'y \in L$, for all $x, y \in A^*$. If L is regular, a well known result states that $\mathcal{M}(L)$ is a finite monoid.

Historically, this notion arose in the context of variable-length codes. We recall that X^* is a free monoid if and only if X is a *code*, i.e. for all $x_1, \dots, x_n, x'_1, \dots, x'_m \in X$, we have

$$x_1 \cdots x_n = x'_1 \cdots x'_m \Leftrightarrow n = m \text{ and } \forall i \in \{1, \dots, n\} x_i = x'_i$$

A remarkable class of codes is the class of *biprefix codes* C . $C \subseteq A^*$ is biprefix if no word in C is a proper prefix or a proper suffix of another element in C , i.e. $C \cap CA^+ = C \cap A^+C = \emptyset$. For instance, uniform codes $A^d, d \geq 1$, are biprefix codes.

Here we report two known results on codes. Theorem 2 has been used to prove Proposition 4. Theorem 3 completely describes finite group codes.

Theorem 2. [3,19] *Let $X \subseteq A^+$ be a code. Then X^* is closed under conjugacy relation if and only if $\mathcal{M}(X^*)$ is a group.*

We explicitly note that, under the hypothesis that X is a regular language, $\mathcal{M}(X^*)$ is a group if and only if X is a *group code*, i.e. a group G and a surjective morphism $\phi: A^* \rightarrow G$ exist such that $X^* = \phi^{-1}(H)$, H being a subgroup of G . Group codes are biprefix codes.

Theorem 3. [3,19] *Let $X \subseteq A^+$ be a group code. X is finite if and only if $X = A^d, d \geq 1$.*

Example 4. [2] Group codes exist which are not finite. Let $A = \{a, b\}$ and $M_a = (b^*(ab^*a)^*)^*$ be the set of words with an even number of a . M_a is generated by $X_a = b \cup ab^*a$, which is a (biprefix) code. Then $M_a = X_a^*$ is a free monoid. Moreover, X_a is a regular group code, being $X_a^* = \phi^{-1}(0)$, where $\phi: A^* \rightarrow \mathbb{Z}/2\mathbb{Z}$ is the morphism given by $\phi(a) = 1, \phi(b) = 0$ and with $\{0\}$ a subgroup of $\mathbb{Z}/2\mathbb{Z}$ (w.r.t. $+$). Observe that for the submonoid $X^* = \{w \in A^* \mid |w|_a = |w|_b\}$ composed of the words in A^* having as many a 's as b 's, we have $X^* = \phi^{-1}(0)$ where $\phi: A^* \rightarrow \mathbb{Z}$ is such that $\phi(a) = 1, \phi(b) = -1$, and with $\{0\}$ a subgroup of \mathbb{Z} (w.r.t. $+$). X is called a *Dyck code* over A and it is a (non regular) group code.

Example 2 (continued). Consider $L_1 = \{w \in A^* \mid \exists h, k \in \mathbb{N} \mid |w|_a = 2k, |w|_b = 2h\}$. Obviously, $L_1 = M_a \cap M_b$. Thus L_1 is a free monoid which is closed under conjugacy, since it is the intersection of two free monoids both closed under conjugacy. We have already observed that L_1 is a regular language. Furthermore, since L_1 is a free monoid, it follows that $L_1 = X^*$, where X is a code [2].

Moreover, it is well known that when X is a code, X^* is regular if and only if X is regular [2]. Since X is a code which is regular and $X^* = L_1$ is closed under conjugacy relation, by Theorem 2 we have that X is a group code. Consequently, $L_1 = X^*$ is a star language and $Cir(L_1) = Cir(X^*)$ is CPA generated, thanks to Corollary 1. A splicing system generating $Cir(L_1)$ is $SC_{PA} = (A, I, R)$ with $I = \{\sim aa, \sim bb, \sim aabb, \sim abab\} \cup 1$ and $R = \{a|1\$1|a, b|1\$1|b, a|1\$1|b\}$.

Remark 2. We can observe that each regular language closed under conjugacy is contained in a star language generated by a regular group code which is minimal w.r.t. inclusion. Indeed, let L be a regular language which is closed under conjugacy. A^* is a free monoid containing L , which is regular and closed under conjugacy. Thus, the intersection of all regular free monoids containing L which are closed under conjugacy is a free monoid containing L , closed under conjugacy (thus generated by a regular group code) and minimal w.r.t. inclusion.

Finally, Proposition 4 shows that the hypothesis contained in Theorem 1 is satisfied by regular group codes.

Proposition 4. *Let X be a regular group code and let \mathcal{A} be a trim deterministic finite state automaton recognizing X^* . For any cycle c in \mathcal{A} , $c \in X^*$.*

Proof. Let X be a regular group code and let \mathcal{A} be a trim deterministic finite state automaton recognizing X^* . Let c be a cycle in \mathcal{A} . Then, $x, y \in A^*$ exist such that $xcy \in X^*$ and $xy \in X^*$. Moreover, $ycx, yx \in X^*$, being X^* closed under conjugacy, and $c \in X^*$, since X^* is biunitary. We recall that X is a biprefix code if and only if X^* is biunitary, i.e. X^* is both left and right unitary, a submonoid X^* being right (resp. left) unitary if, for all $u, v \in A^*$, $u, uv \in X^*$ (resp. $u, vu \in X^*$) implies $v \in X^*$ [2]. \square

Corollary 1. *For each regular group code X , $Cir(X^*)$ is a circular splicing language.*

As we have already said, we prove that all star languages X^* with X a finite set have their circularization in $C(Fin, Fin)$. In the proof of this result, we also give a splicing system generating $Cir(X^*)$ whose construction is simpler than the one given in the proof of Theorem 1.

Proposition 5. *For each finite set X , $Cir(X^*)$ is a circular splicing language.*

Proof. It is clear that the splicing system $SC_{PA} = (A, I, R)$ generates $Cir(X^*)$, when we choose $I = \{\sim x \mid x \in X\} \cup 1$ and $R = \{x_i|1\$1|x_j \mid x_i, x_j \in X\}$.

4 Conclusions and Future Work

As we have already said in Section 1, additional hypotheses can be added to the definitions of circular splicing given by Paun and Pixton. In this section we report the result already obtained which uses the hypotheses recalled below.

Hypothesis 1. R is a symmetric scheme, i.e. for any rule $r = u_1|u_2\$u_3|u_4$ (resp. $r = (\alpha, \alpha'; \beta)$) in a splicing system SC_{PA} (resp. SC_{PI}) there is the rule $\bar{r} = u_3|u_4\$u_1|u_2$ (resp. $\bar{r} = (\alpha', \alpha; \beta')$).

Remark 3. Any set R of rules in a splicing system SC_{PI} is implicitly supposed to be symmetric.

Hypothesis 2. R is a reflexive scheme, i.e. for any rule $u_1|u_2\$u_3|u_4$ (resp. $(\alpha, \alpha'; \beta)$) in a splicing system SC_{PA} (resp. SC_{PI}) there is the rule $u_1|u_2\$u_1|u_2$ (resp. $(\alpha, \alpha; \alpha)$).

Hypothesis 3. Self-splicing. Self-splicing is defined in a splicing system SC_{PA} (resp. SC_{PI}) producing $\sim u_4xu_1$ and $\sim u_2yu_3$ from $\sim xu_1u_2yu_3u_4$ and the rule $u_1|u_2\$u_3|u_4$ (resp. $\sim \beta\epsilon', \sim \beta'\epsilon$ starting from $\sim \alpha\epsilon\alpha'\epsilon'$ and r, \bar{r} as above).

Problem 3 generalizes Problem 2 when we take into account the additional hypotheses given above.

Problem 3. Can we characterize $F \cap C(\text{Fin}, \text{Fin})$ (resp. $F \cap C(\text{Reg}, \text{Fin})$) for each class F of languages in the Chomsky hierarchy, for every definition of $C(\text{Fin}, \text{Fin})$ (resp. $C(\text{Reg}, \text{Fin})$) and for every possible combination of the three hypotheses above?

Below we report a known result which uses Pixton's systems with hypotheses 1, 2 and which generalizes a similar theorem proved for linear splicing [18].

Theorem 4. [18] *Let $SC_{PI} = (A, I, R)$ be a circular splicing system with I a circular regular language and R reflexive and symmetric. Then $C = C(SC_{PI})$ is regular.*

We conclude this section with some remarks related to Problem 2, i.e. the characterization of the circular regular languages generated by finite circular splicing, and with some problems which are still open.

First of all, we point out that rational languages exist which are not star languages and which are the full linearization of regular circular splicing languages. Indeed, $\sim(ab)^*$ has $(ab)^* \cup (ba)^*$ as full linearization, and this is not a star language, but it is generated by $SC_{PA} = (A, I, R)$, where $A = \{a, b\}$, $I = \{\sim ab\}$, $R = \{ab|1\$1|ab\}$.

Secondly, note that the class of regular languages closed under conjugacy relation is closed under \cup . However, while $\text{Cir}((A^2)^*)$, $\text{Cir}((A^3)^*) \in C(\text{Fin}, \text{Fin})$, as shown in Section 3, we have $\text{Cir}((A^2)^*) \cup \text{Cir}((A^3)^*) \notin C(\text{Fin}, \text{Fin})$. This means that $\text{Reg} \cap C(\text{Fin}, \text{Fin})$ is not closed w.r.t. \cup . One could ask for additional hypotheses to be added so that the union of circular regular splicing languages will still be a circular splicing language. This research line could be a promising direction towards a complete characterization of the class of circular regular languages that are generated by finite circular splicing. In this direction, further investigations are contained in a work in progress [5].

Another interesting problem could be the investigation of the *splicing subsystem* of a given circular splicing system. For example, $\text{Cir}((A^2)^*)$, for $A = \{a, b\}$, is CPA generated by $I = \sim A^2$ and $R' = \{w_1|1\$1|w_2 \mid w_1, w_2 \in A^2\}$ (Proposition 5). Thus, by extracting $R_1 = \{aa|1\$1|aa\}$, we obtain $\sim(aa)^*$, with $R_2 = \{bb|1\$1|bb\}$ we generate $\sim(bb)^*$ and with $R_3 = \{ab|1\$1|ab\}$ we obtain $\sim(ab)^*$. It could be interesting to prove or to disprove that circular languages can

be obtained by extraction as described before. Let us take $L_1 = \{w \in A^* \mid \exists h, k \in \mathbb{N} \mid |w|_a = 2k, |w|_b = 2h\}$ as in Example 2. The circular language $Cir(L_1)$ cannot be generated by choosing a subset of $R = \{a|1\$1|a, b|1\$1|b, a|1\$1|b\}$. This investigation is related to the notion of a *minimal splicing system* introduced in [17], where it was called *descriptive complexity*. It goes without saying that this notion is the counterpart of the minimal automaton for regular languages. Here, the minimality of the system could be referred to the cardinality of R or to the length of the rules in R .

Acknowledgments. Many thanks to J. Berstel and A. Restivo for suggesting us references [3] and [19] during the conference Words99 [6]. The authors also wish thank A. Bertoni for useful discussions.

References

1. L. M. Adleman, *Molecular computation of solutions to combinatorial problems*, Science, 226, (1994), 1021 – 1024.
2. J. Berstel, D. Perrin, *Theory of codes*, Academic Press, New York, (1985).
3. J. Berstel, A. Restivo, *Codes et sousmonoides fermes par conjugaison*, Sem. LITP, 81 – 45, (1981), 10 pages.
4. P. Bonizzoni, R. Zizza, *Deciding whether a regular language is a splicing language*, submitted, (1999).
5. P. Bonizzoni, C. De Felice, G. Mauri, R. Zizza, *Circular splicing and regular languages*, manuscript, (2000).
6. P. Bonizzoni, C. De Felice, G. Mauri, R. Zizza, *Linear and circular splicing*, WORDS99, (1999).
7. P. Bonizzoni, C. Ferretti, G. Mauri, R. Zizza, *Separating some splicing models*, Grammar Systems 2000, (2000).
8. K. Culik, T. Harju, *Splicing semigroups of dominoes and DNA*, Discrete Appl. Math., 31, (1991), 261 – 277.
9. R. W. Gatterdam, *Algorithms for splicing systems*, SIAM Journal of Computing, 21 : 3, (1992), 507 – 520.
10. D. Giammarresi, A. Restivo, *Two-dimensional Languages*, in: Handbook of Formal Languages, G. Rozenberg & A. Salomaa, Eds., Springer Verlag, Vol. 3, (1996), 215 – 267.
11. T. Head, *Formal Language Theory and DNA: an analysis of the generative capacity of specific recombinant behaviors*, Bull. Math. Biol., 49, No. 5, (1987), 737 – 759.
12. T. Head, Gh. Paun, D. Pixton, *Language theory and molecular genetics: generative mechanisms suggested by DNA recombination*, in: Handbook of Formal Languages, G. Rozenberg & A. Salomaa, Eds., Springer Verlag, Vol. 2, (1996), 295 – 360.
13. J.E. Hopcroft, J.D. Ullman, *Introduction to Automata Theory, Languages, and Computing*, Addison-Wesley, Reading, Mass. (1979).
14. S.M. Kim, *Computational modeling for genetic splicing systems*, SIAM Journal of Computing, 26, (1997), 1284 – 1309.
15. M. Lothaire, *Combinatorics on Words*, Encyclopedia of Math. and its Appl., Addison Wesley Publishing Company (1983).
16. G. Paun, *On the splicing operation*, Discrete Applied Math., 70, (1996), 57 – 79.

17. G. Paun, G. Rozenberg, A. Salomaa, *DNA computing, New Computing Paradigms*, Springer-Verlag, (1998).
18. D. Pixton, *Regularity of splicing languages*, Discrete Applied Math. 69, (1996), 101 – 124.
19. C. Reis, G. Thierren, *Reflective star languages and codes*, Information and Control, 42, (1979), 1 – 9.
20. R. Siromoney, K.G. Subramanian, A. Dare, *Circular DNA and Splicing Systems*, Proc. of ICPIA, LNCS 654, Springer-Verlag, (1992), 260 – 273.

Molecular Computing with Generalized Homogeneous P-Systems

Rudolf Freund¹ and Franziska Freund²

¹ Institut für Computersprachen
Technische Universität Wien
Karlsplatz 13

A-1040 Wien, Austria
tel.: ++43 1 58801 18542
`rudi@logic.at`

² De La Salle Schools
Strebersdorf
Anton Böck-Gasse 37
A-1215 Wien, Austria
tel.: ++43 1 58801 18542
`ffreund@logic.at`

Abstract. Recently P-systems were introduced by Gheorghe Păun as a new model for computations based on membrane structures. The basic variants of P-systems shown to have universal computational power only took account of the multiplicities of atomic objects, some other variants considered rewriting rules on strings. Using the membranes as a kind of filter for specific objects when transferring them into an inner compartment or out into the surrounding compartment turned out to be a very powerful mechanism in combination with suitable rules to be applied within the membranes in the model of generalized P-systems, GP-systems for short. GP-systems were shown to allow for the simulation of graph controlled grammars of arbitrary type based on productions working on single objects; moreover, various variants of GP-systems using splicing or cutting and recombination of strings were shown to have universal computational power, too. In this paper, we consider GP-systems with homogeneous membrane structures, GhP-systems for short, using splicing or cutting and recombination of string objects with specific markers at the ends of the strings that can be interpreted as electrical charges. The sum of these electrical charges determines the permeability of the membranes to the string objects, and we allow only objects with the absolute value of the difference of electrical charges being equal to 1 to pass a membrane in both directions. We show that such GhP-systems have universal computational power; for GhP-systems using splicing and a bounded number of markers the obtained results are optimal with respect to the underlying membrane structure. Moreover, a very restricted variant of such GhP-systems characterizes the (strictly) minimal linear languages.

1 Introduction

In the model of P-systems – as introduced by Gheorghe Păun in [10] – the most important feature is the membrane structure (for a chemical variant of this idea see [1]) consisting of membranes hierarchically embedded in the outermost *skin* membrane. Every membrane encloses a *region* possibly containing other membranes; the part delimited by the membrane labelled by k and its inner membranes is called *compartment* k . A region delimited by a membrane not only may enclose other membranes but also specific objects and operators, which in general are considered as multisets, as well as evolution rules, which in *generalized P-systems* (*GP-systems*) as introduced in [4] and [5] are evolution rules for the operators. In GP-systems, ground operators as well as transfer operators (simple rules of that kind are called travelling rules in [17]) are taken into account; these transfer operators transfer objects or operators (or even rules) either to the outer compartment or to an inner compartment delimited by a membrane of specific kind with also checking for some permitting and/or forbidding conditions on the objects to be transferred. In that way, the membranes act as a filter like in test tube systems (e.g., see [13] and [6]). In [5] it was shown how GP-systems with splicing or cutting and recombination rules can simulate test tube systems using the corresponding type of molecular rules. In contrast to the original definition of P-systems (e.g., see [2], [9]), in GP-systems no priority relations on the rules are used, and we do not enforce parallelism guarded by a universal clock.

Already in his first papers on membrane computing ([9]), Gheorghe Păun considered P-systems using the splicing operation on strings. P-systems based on splicing further were investigated in [15] and [16]. In all these variants, one main feature was that together with the application of a splicing rule the resulting string(s) actively could be moved out of a region or moved into an inner membrane. In [8] a specific model of generalized P-systems based on cutting and recombination with special homogeneous membrane structures were considered, the objects (assumed to be available in an unbounded number) being able to pass the membranes at a specific depth of the membrane structure depending on their electrical charges only.

In this paper we consider such GhP-systems (generalized homogeneous P-systems) based on splicing or cutting and recombination with a completely uniform permeability condition for all the membranes in the system, i.e., only objects with the absolute value of the difference of electrical charges being equal to 1 can pass a membrane in both directions. For GhP-systems based on splicing, the simplest non-trivial membrane structure is already sufficient for obtaining universal computational power. On the other hand, for GhP-systems based on cutting and recombination the optimality of the obtained results remains as an open problem.

Both the membrane structure as well as the operations used in GhP-systems based on splicing or cutting and recombination are motivated by nature, and moreover, the uniform permeability conditions for the membranes we are going to use in this paper look quite realistic. Yet despite this chemically/biologically motivated background, real implementations of GhP-systems - like implemen-

tations of other variants of P-systems - in the lab (“in vitro”) remain a major challenge for the future.

In the following section we start with some preliminary notions from formal language theory and then give a general definition of a molecular system that also captures the notion of splicing and cutting/recombination systems of strings. In the third section we introduce the model of GhP-systems considered in this paper; as a first result, we show how a very restricted variant of GhP-systems characterizes the strictly minimal linear languages; moreover, we investigate the computational power of GhP-systems with the simplest membrane structure of depth zero. Our main result showing that GhP-systems using splicing or cutting/recombination have universal computational power is elaborated in the fourth section. A short summary and an outlook to future research topics and open problems conclude the paper.

2 Preliminary Definitions

For an alphabet V , by V^* we denote the free monoid generated by V under the operation of concatenation; the *empty string* is denoted by λ , and $|w|$ denotes the length of the string w , $w \in V^*$. $V^* \setminus \{\lambda\}$ is denoted by V^+ ; any subset of V^* (V^+) is called a (λ -free) *language*. \mathbf{Z} denotes the set of integers; \mathbf{N} denotes the set of non-negative integers.

A *minimal linear grammar* G is a quadruple $(\{S\}, V_T, P, S)$, where S is the only non-terminal symbol and the start symbol of the grammar, V_T is the terminal alphabet, and P is the set of linear productions of the forms $S \rightarrow w$ with $w \in V_T^*$ or $S \rightarrow uSw$ with $uw \in V_T^+$. If, moreover, $S \rightarrow \lambda$ is the only production in P of the form $S \rightarrow w$ with $w \in V_T^*$, then G is called a *strictly minimal linear grammar*.

A *molecular system* is a quadruple $\sigma = (B, B_T, P, A)$, where B and B_T are sets of *objects* and *terminal objects*, respectively, with $B_T \subseteq B$, P is a set of *productions*, and A is a set of axioms from B . A production p in P in general is a partial recursive relation $\subseteq B^k \times B^m$ for some $k, m \geq 1$, where we also demand that the domain of p is recursive (i.e., given $w \in B^k$ it is decidable if there exists some $v \in B^m$ with $(w, v) \in p$) and, moreover, that the range for every w is finite, i.e., for every $w \in B^k$, $\text{card}(\{v \in B^m \mid (w, v) \in p\}) < \infty$. For any two sets L and L' over B , we say that L' is computable from L by a production p if and only if for some $(w_1, \dots, w_k) \in B^k$ and $(v_1, \dots, v_m) \in B^m$ with $(w_1, \dots, w_k, v_1, \dots, v_m) \in p$ we have $\{w_1, \dots, w_k\} \subseteq L$ and $L' = L \cup \{v_1, \dots, v_m\}$; we also write $L \Rightarrow_p L'$ and $L \Rightarrow_\sigma L'$. A computation in σ is a sequence L_0, \dots, L_n such that $L_i \subseteq B$, $0 \leq i \leq n$, $n \geq 0$, as well as $L_i \Rightarrow_\sigma L_{i+1}$, $0 \leq i < n$; in this case we also write $L_0 \Rightarrow_\sigma^n L_n$, and moreover, we write $L_0 \Rightarrow_\sigma^* L_n$ if $L_0 \Rightarrow_\sigma^n L_n$ for some $n \geq 0$. The *language generated by* σ is

$$L(\sigma) = \{w \mid A \Rightarrow_\sigma^* L, w \in L \cap B_T\}.$$

The special productions on string objects we shall consider in the following are the cutting and recombination operations as well as the splicing operation:

A *cutting/recombination scheme* (a *CR-scheme* for short) is a quadruple (V, M, C, R) , where V is a finite alphabet; M is a finite set of *markers*; V and M are disjoint sets; C is a set of *cutting rules* of the form $u\#l\$m\#v$, where $u \in MV^* \cup V^*$, $v \in V^*M \cup V^*$, and $m, l \in M$, and $\#, \$$ are special symbols not in $V \cup M$; $R \subseteq M \times M$ is the recombination relation representing the *recombination rules*. Cutting and recombination rules are applied to objects from MV^*M . For $x, y, z \in MV^*M$ and a cutting rule $c = u\#l\$m\#v$ we define $x \Rightarrow_c (y, z)$ if and only if for some $\alpha \in MV^*$ and $\beta \in V^*M$ we have $x = \alpha uv\beta$ and $y = \alpha ul$, $z = mv\beta$. For $x, y, z \in MV^*M$ and a recombination rule $r = (l, m)$ from R we define $(x, y) \Rightarrow_r z$ if and only if for some $\alpha \in MV^*$ and $\beta \in V^*M$ we have $x = \alpha l$, $y = m\beta$, and $z = \alpha\beta$. For a CR-scheme $\sigma = (V, M, C, R)$ and any language $L \subseteq MV^*M$, $\sigma(L)$ then denotes the set of all objects obtained by applying one cutting or one recombination rule to objects from L . We also define $\sigma^0(L) = L$ and $\sigma^{i+1}(L) = \sigma(\sigma^i(L))$ for all $i \geq 0$, as well as $\sigma^{(0)}(L) = L$ and $\sigma^{(i+1)}(L) = \sigma^{(i)}(L) \cup \sigma(\sigma^{(i)}(L))$ for all $i \geq 0$; moreover, we denote $\sigma^{(*)}(L) = \bigcup_{i=0}^{\infty} \sigma^{(i)}(L)$. An *extended CR-system* is a molecular system of type *CR* σ , $\sigma = (MV^*M, M_T V_T^* M_T, P, A)$, where $V_T \subseteq V$ is the set of terminal symbols, $M_T \subseteq M$ is the set of terminal markers, A is the set of axioms, P is the union of the relations (productions) defined by the cutting rules from C ($\subseteq (MV^*M) \times (MV^*M)^2$) and the recombination rules from R ($\subseteq (MV^*M)^2 \times (MV^*M)$), and (V, M, C, R) is the underlying CR-scheme.

Throughout this paper we shall restrict ourselves to markers that can be interpreted as electrical charges of ions, i.e., we shall write $[+k]$ and $[-k]$, $k \in \mathbf{N}$, for these special markers. In that sense, the recombination rules we use will be of the simple forms $([+k], [-k])$ and $([-k], [+k])$, $k \in \mathbf{N}$; the objects we are working with are of the form $[+k]w[-l]$ with $w \in V^*$. On such objects from $\mathbf{Z}'V^*\mathbf{Z}'$, where \mathbf{Z}' is a finite subset of $\{[-k], [+k] \mid k \in \mathbf{N}\}$ (i.e., on linear objects with electrical charges at both ends) we can also define the splicing operation in the following way:

Let \mathbf{Z}' be a finite subset of $\{[-k], [+k] \mid k \in \mathbf{N}\}$ and let V be a finite alphabet. An *extended splicing system* (*extended H system*) over $\mathbf{Z}'V^*\mathbf{Z}'$ is a molecular system of type *H* σ , $\sigma = (\mathbf{Z}'V^*\mathbf{Z}', \mathbf{Z}''V_T^*\mathbf{Z}'', P, A)$, where $V_T \subseteq V$ is the set of terminal symbols, $\mathbf{Z}'' \subseteq \mathbf{Z}'$ is the set of terminal markers, A is the set of axioms, P is a set of splicing rules of the form $u_1\#u_2\$v_1\#v_2$, where $u_1, v_1 \in \mathbf{Z}'V^* \cup V^*$, $u_2, v_2 \in V^*\mathbf{Z}' \cup V^*$, and $\#, \$$ are special symbols not in $V \cup \mathbf{Z}'$; for $x, y, z \in \mathbf{Z}'V^*\mathbf{Z}'$ and a splicing rule $s = u_1\#u_2\$v_1\#v_2$ we define $(x, y) \Rightarrow_s z$ if and only if for some $x_1, y_1 \in \mathbf{Z}'V^*$ and $x_2, y_2 \in V^*\mathbf{Z}'$ we have $x = x_1u_1u_2x_2$ and $y = y_1v_1v_2y_2$ as well as $z = x_1u_1v_2y_2$ (we omit the second result $y_1v_1u_2x_2$).

3 Generalized Homogeneous P-Systems (GhP-Systems)

In this section we quite informally describe the model of GhP-systems discussed in this paper, especially the features not captured by the original model of P-systems as described in [2], [9], and [10]. In these papers, only the number of

symbols is counted in the multiset sense, whereas in [14] at least the outputs are strings. In generalized P-systems (for the basic definition of GP-systems the reader is referred to [4] and [5]) the objects usually are strings or graphs, etc.

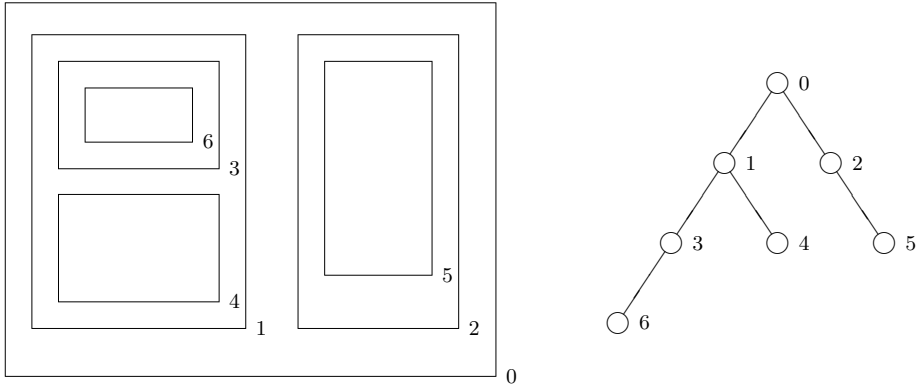


Fig. 1. Membrane structure $[_0[1[3[6]6]3[4]4]_1[2[5]5]_2]_0$.

The basic ingredient of a $(G(h))P$ -system is a *membrane structure* consisting of several membranes placed within one unique surrounding membrane, the so-called skin membrane. All the membranes can be labelled (in a one-to-one manner) by natural numbers; the outermost membrane (skin membrane) always is labelled by 0. In that way, a membrane structure can uniquely be described by a string of correctly matching parentheses, where each pair corresponds to a membrane. For example, the membrane structure depicted in Figure 1, which within the skin membrane contains two inner membranes labelled by 1 - containing membrane 3 (with membrane 6 inside) and membrane 4 - as well as by 2 (containing membrane 5) is described by $[_0[1[3[6]6]3[4]4]_1[2[5]5]_2]_0$. Figure 1 also shows that a membrane structure graphically can be represented by a Venn diagram, where two sets can either be disjoint or one set be the subset of the other one. In this representation, every membrane encloses a *region* possibly containing other membranes; the part delimited by the membrane labelled by k and its inner membranes is called *compartment k* in the following. The space outside the skin membrane is called *outer region*. Another obvious representation of a membrane structure is a tree as shown in Figure 1; in that sense the depth of a membrane structure can be defined as the depth of the corresponding tree; e.g., the depth of $[_0[1[3[6]6]3[4]4]_1[2[5]5]_2]_0$ is 3.

Informally, in [9] and [10] *P-systems* were defined as membrane structures containing multisets of objects in the compartments k as well as evolution rules for the objects. A priority relation on the evolution rules guarded the applica-

tion of the evolution rules to the objects, which had to be affected in parallel (if possible according to the priority relation). The output was obtained in a designated compartment from a halting configuration (i.e., a configuration of the system where no rules can be applied any more).

A *generalized homogeneous P-system (GhP-system)* of molecular type X is a construct γ , $\gamma = (B, B_T, P, A, \mu, I, in, out)$, where

- (B, B_T, P, A) is a molecular system of type X (we shall consider molecular systems of type CR and H in the following);
- μ is a membrane structure (with the membranes labelled by natural numbers $0, \dots, n$);
- $I = (I_0, \dots, I_n)$, where I_k is the initial contents of compartment k containing a set of objects from A as well as a set of rules from P ;
- in for each $j \in \{0, 1, \dots, n\}$ specifies a condition an object must fulfill in order to be able to pass into the inner compartment of a membrane k , $k \in \{1, \dots, n\}$, in the region enclosed by membrane j ;
- out specifies a condition an object must fulfill in order to be able to pass a membrane j , $j \in \{0, 1, \dots, n\}$, into its surrounding compartment.

A *computation* in γ starts with the initial configuration with I_k being the contents of compartment k . In contrast to the original definition of P-systems, the objects in the compartments are not treated in the multiset sense; instead we assume all objects occurring in a compartment to be available in an unbounded number. A transition from one configuration to another one is performed by applying a rule (from P) in I_k to objects in compartment k or by moving an element out of a compartment k or into a compartment k according to the conditions given by out and in . The language generated by γ is the set of all terminal objects $w \in B_T$ obtained in the outer region by some computation in γ .

In test tube systems (e.g., see [6]), the contents of the tubes is redistributed to all other tubes according to specific input or output filters; the operations in the tubes are based on molecular systems of type H or CR . In [6] we showed that test tube systems with only two test tubes using splicing or cutting and recombination rules in the tubes and filters of a special type between the tubes have the computational power of arbitrary grammars and Turing machines, respectively. According to the definition given above, in the GhP-systems of type CR used in [8], we only allowed one unique out-condition, whereas the in-conditions specified by in depended on the region the membrane to be passed lay in, where the in-conditions of regions at the same level of the membrane structure coincided; GhP-systems of type CR with a membrane structure of the form $[0[1[n+1]_{n+1}]_1 \dots [n[2n]_{2n}]_n]_0$ of depth two were shown to have universal computational power in [8].

In this paper we will use a completely uniform permeability condition for the membranes, i.e., only objects with the absolute value of the sum of electrical charges being equal to 1, can pass the membranes in both directions; hence in the following, we shall not specify this uniform static in- and out-conditions any

more. Although using this special permeability condition for the membranes, we can improve the results for GhP-systems of type CR shown in [8], i.e., we can reduce the complexity of the membrane structure to depth one, and for GhP-systems of type H we can establish even optimal results, i.e., even with the simplest non-trivial membrane structure $[0[1]_1]_0$ we obtain universal computational power (compare with the results proved in [5], [15], and [16]).

To illustrate the model of GhP-systems, we give an example of a GhP-system of type CR generating the (non-regular) language $\{a^n b^n \mid n \geq 0\}$:

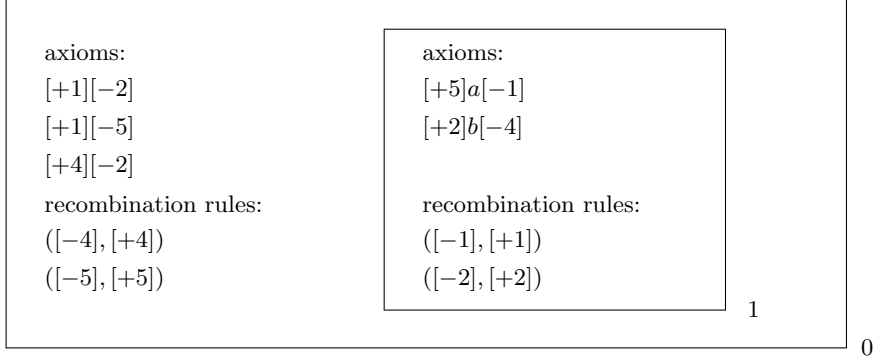


Fig. 2. GhP-system generating $\{a^n b^n \mid n \geq 0\}$.

Example 1. The main ingredients of a GhP-system of type CR generating the language $\{a^n b^n \mid n \geq 0\}$ are depicted in Figure 2. In the skin membrane we start with (terminal) objects of the form $[+1]a^n b^n [-2]$ for some $n \geq 0$; these objects can enter compartment 1, where one application of each of the recombination rules $([-1], [+1])$ and $([-2], [+2])$ with the corresponding axioms yields $[+5]a^{n+1}b^{n+1}[-4]$, which object can pass back to compartment 0, where the application of the recombination rules $([-4], [+4])$ and $([-5], [+5])$ yields the terminal object $[+1]a^{n+1}b^{n+1}[-2]$. \square

Replacing the recombination rules $([-k], [+k])$, $k \in \{1, 2, 4, 5\}$, by the corresponding special splicing rules $\#[-k]\$[+k]\#$ yields a GhP-system of type H generating $\{a^n b^n \mid n \geq 0\}$.

In a similar way as it was shown for the language $\{a^n b^n \mid n \geq 0\}$ in Example 1, all strictly minimal linear languages can be generated by GhP-system of type CR using only recombination rules in a membrane structure of depth 1:

Lemma 2. *Any strictly minimal linear language generated by a strictly minimal linear grammar G , $G = (\{S\}, V_T, P, S)$, can be generated by a GhP-system of type CR using only recombination rules within the membrane structure $[0[1]_1 \dots [n]_n]_0$, where n is the number of linear productions of the form $S \rightarrow uSw$ with $uw \in V_T^+$ in P .*

Proof. The main parts of the GhP-system are depicted in Figure 3. The terminal objects $w \in L(G)$ can leave compartment 0 in the form $[+1]w[-2]$. \square

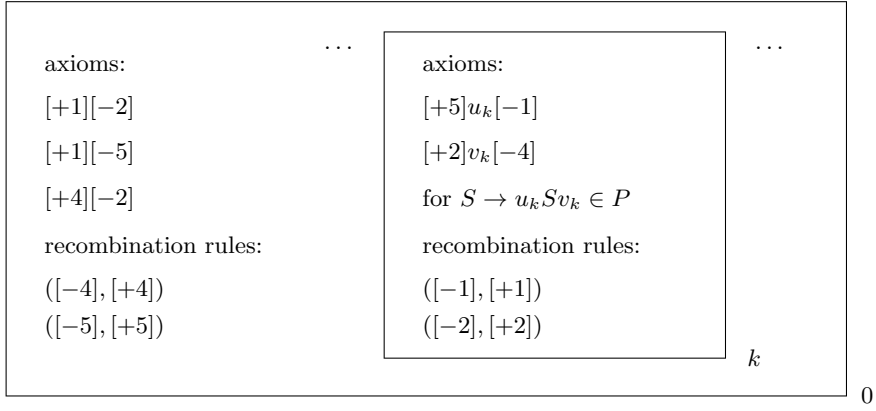


Fig. 3. GhP-system of type CR generating $L(G)$, where $G = (\{S\}, V_T, P, S)$ is a strictly minimal linear grammar.

Without further proof we mention that GhP-systems of type CR of the very restricted form as constructed in the preceding lemma exactly characterize the strictly minimal linear languages. Moreover, as already mentioned above, the recombination rule $([-k], [+k])$ can be simulated by the splicing rule $\#[-k]\$[+k]\#$, hence, in that way we also obtain a characterization of strictly minimal linear languages by GhP-systems of type H .

To the end of this section, we investigate the computational power of GhP-systems with the simplest membrane structure $[_0]_0$.

Lemma 3. *The family of regular languages can be characterized by GhP-systems of type H/CR with the simplest membrane structure $[_0]_0$.*

Proof. Extended H-systems as well as extended CR-systems exactly characterize the family of regular languages (see [13] and [18]). Hence, if we consider a GhP-system of type H/CR with the simplest membrane structure $[_0]_0$, then the resulting language in compartment 0 is regular. As the family of regular languages is closed under intersection, the language obtained in the outer region as the intersection of this language in compartment 0 and a regular language of the form $\mathbf{Z}''V_T^*\mathbf{Z}''$ with \mathbf{Z}'' being a finite subset of $\{[-k], [+k] \mid k \in \mathbf{N}\}$ again is regular.

On the other hand, let $G, G = (V_N, V_T, P, S)$, be a regular grammar with productions in P of the form $B \rightarrow cD$ or $B \rightarrow c$, $B, D \in V_N$, $c \in V_T$:

$L(G)$ is generated by the GhP-system γ of type H :

$$\begin{aligned}
\gamma &= (B, B_T, R, A, [0]_0, (A \cup R)) \\
B &= \mathbf{Z}' V^* \mathbf{Z}' \\
\mathbf{Z}' &= \{[+k], [-k] \mid 1 \leq k \leq 4\} \\
V &= V_N \cup V_T \\
B_T &= \mathbf{Z}'' V_T^* \mathbf{Z}'' \\
\mathbf{Z}'' &= \{[+k], [-k] \mid 1 \leq k \leq 2\} \\
R &= R_1 \cup R_2 \\
R_1 &= \{u \# B[-4] \$ [+4] \# cD[-4] \mid B \rightarrow cD \in P, u \in W\} \\
R_2 &= \{u \# B[-4] \$ [+4] \# c[-1] \mid B \rightarrow c \in P, u \in W\} \\
W &= \{[+1]\} \cup \{[+1]\} V_T \cup V_T^2 \\
A &= \{[+1] S[-4]\} \cup A_1 \cup A_2 \\
A_1 &= \{[+4] cD[-4] \mid B \rightarrow cD \in P\} \\
A_2 &= \{[+4] c[-2] \mid B \rightarrow c \in P\}
\end{aligned}$$

The application of a production $B \rightarrow cD \in P$ is simulated by using a suitable splicing rule of the form $u \# B[-4] \$ [+4] \# cD[-4]$ together with the axiom $[+4] cD[-4]$, the application of a terminal production $B \rightarrow c \in P$ is simulated by using a suitable splicing rule of the form $u \# B[-4] \$ [+4] \# c[-2]$ together with the axiom $[+4] c[-2]$, thus finally yielding a terminal object $[+1] w[-2]$ for some $w \in V_T^+$.

A GhP-system γ of type CR , $\gamma = (B, B_T, R, A, [0]_0, (A \cup R))$, generating $L(G)$ is more complicated, because in this case we have to store the information about the non-terminal symbol at the end of the sentential form in the marker on the right-hand side; on the other hand, the reader should observe that we only need recombination rules, but no cutting rules:

Let $V_N = \{X_{2i} \mid 2 \leq i \leq m\}$ and $S = X_4$.

Then we take $B = \mathbf{Z}' V^* \mathbf{Z}'$, $\mathbf{Z}' = \{[+k], [-k] \mid 1 \leq k \leq 2m\}$, and V and B_T as above; moreover, we now take the following recombination rules and axioms:

$$\begin{aligned}
R &= \{([-2i], [+2i]) \mid 2 \leq i \leq m\} \\
A &= \{[+1] [-4]\} \cup A_1 \cup A_2 \\
A_1 &= \{[+2i] c[-2j] \mid X_{2i} \rightarrow cX_{2j} \in P\} \\
A_2 &= \{[+2i] c[-2] \mid X_{2i} \rightarrow c \in P\}
\end{aligned}$$

It is easy to see that a terminal object $[+1] w[-2]$ for some $w \in V_T^+$ is generated in compartment 0 if and only if $w \in L(G)$. \square

In the proofs of Lemma 3 we could also take $B_T = B$, because the filtering out of the terminal objects could be achieved by the permeability condition for the skin membrane 0 only.

In contrast to the rather obvious result established for GhP-systems of type H in Lemma 3, where we only need a limited number of markers, the number of markers is not bounded in the GhP-systems of type CR and depends on the number of non-terminal symbols in the underlying regular grammar, which fact

may give rise to an infinite non-collapsing hierarchy with respect to the number of markers.

4 The Computational Power of GhP-Systems of Type H and CR

The main result we prove in the following establishes the universal computational power of GhP-systems. First we improve the result established for GhP-systems of type CR in [8]:

Theorem 4. *Any recursively enumerable language L can be generated by a GhP-system γ of type CR with a fixed number of markers in the membrane structure $[0]_1[1]\dots[n]_n[0]$.*

Proof. The proof idea to simulate the productions of a grammar generating L like in Post systems in normal form (“rotate-and-simulate”) has already been used in many papers on splicing systems and cutting/recombination systems (see [13] and [6]). Moreover, instead of a grammar G' generating L , we consider a grammar G , $G = (V_N \cup \{B\}, V_T \cup \{d\}, P, S)$, generating the language $L\{d\}$, where the end marker d , $d \notin W$, $W = V_N \cup \{B\} \cup V_T$, in any derivation of a word $w'd$, for $w' \in L$, is generated exactly in the last step of this derivation in G and for each symbol $X \in W$ the production $X \rightarrow X$ is in P . A string $w \in (V_N \cup V_T)^*$ appearing in a derivation of such a grammar G generating $L\{d\}$, is represented by its rotated versions $[+3]Xw_2Bw_1Y[-4]$, where $w = w_1w_2$ and B is a special symbol indicating the beginning of the string within the rotated versions and X, Y are special symbols marking the ends of the strings. A final string first appears in the form $[+7]XdBw'Y[-6]$, where w' is the final result from L which we want to get, and finally leaves compartment 0 in the form $[+1]w'[-2]$.

The GhP-system γ we construct, for each production $p_k : \alpha_k \rightarrow \beta_k$, $1 \leq k \leq n$, where without loss of generality we assume $1 \leq |\alpha_k| \leq 2$, $0 \leq |\beta_k| \leq 2$, and $||\alpha_k| - |\beta_k|| \leq 1$, contains compartment k :

For simulating p_k in compartment k , $1 \leq k \leq n$, we use the *cutting rules*

$u\#[-3]\$[+3]\#\alpha_kY[-6]$, $u \in W \cup \{X\}$, and
 $[+5]X\#[-11]\$[+11]\#v$, $v \in W$,

as well as the *recombination rules*

$([-3], [+3])$ and
 $([-11], [+11])$

together with the axioms

$[+3]Y[-8]$ and
 $[+9]X\beta_k[-11]$.

Thus we obtain $[+9]X\beta_kwY[-8]$ from $[+5]Xw\alpha_kY[-6]$.

In compartment 0 we start with the axiom $[+5]XBSY[-6]$. The axioms $[+5][-9]$ and $[+8][-6]$ together with the recombination rules $([-8], [+8])$ and $([-9], [+9])$ allow us to obtain $[+5]X\beta_kwY[-6]$ from $[+9]X\beta_kwY[-8]$ thus finishing the simulation of p_k . An object of the form $[+9]XvY[-8]$ could also pass immediately into each of the compartments k , $1 \leq k \leq n$, but there it

could not be processed, and therefore the only possibility would be to leave compartment k unchanged again.

Final strings first appearing in the form $[+9] XdBw'Y [-8]$, $w' \in L$, can leave compartment 0 in the form $[+1] w' [-2]$ after the application of suitable cutting rules:

$$\begin{aligned} [+9] XdB\# [-1] \$ [+1] \#a, a \in V_T \cup \{Y\}, \text{ and} \\ b\# [-2] \$ [+2] \#Y [-8], b \in V_T \cup \{[+1]\}. \end{aligned}$$

If a cutting rule $b\# [-2] \$ [+2] \#Y [-8]$ is applied too early, we obtain an object $[+5] Xau [-2]$ with $a \neq d$, which object cannot be processed any more. A terminal object $[+1] w' [-2]$ not only can leave compartment 0, but also enter any other compartment k , $1 \leq k \leq n$, yet there a terminal object cannot be processed any more.

According to the definitions and explanations above, we obtain the following formal description of the GhP-system γ :

$$\begin{aligned} \gamma &= (B, B_T, R, A, [{}_0[1]_1 \dots [{}_n]_n]_0, (A_0 \cup R_0, A_1 \cup R_1, \dots, A_n \cup R_n)) \\ B &= \mathbf{Z}' V^* \mathbf{Z}' \\ \mathbf{Z}' &= \{[+k], [-k] \mid 1 \leq k \leq 11\} \\ V &= V_N \cup V_T \cup \{d, B, X\} = W \cup \{d, X\} \\ B_T &= \mathbf{Z}'' V_T^* \mathbf{Z}'' \\ \mathbf{Z}'' &= \{[+k], [-k] \mid 1 \leq k \leq 2\} \\ R &= R_0 \cup R_1 \cup \dots R_n \\ A &= A_0 \cup A_1 \cup \dots A_n \end{aligned}$$

The sets of axioms are:

$$\begin{aligned} A_0 &= \{[+5] XBSY [-6], [+5] [-9], [+8] [-6]\} \\ A_k &= \{[+3] Y [-8], [+9] X\beta_k [-11]\} \text{ for } 1 \leq k \leq n \end{aligned}$$

The sets of rules R_0, \dots, R_n are specified by the following sets of cutting rules and recombination rules:

$$\begin{aligned} R_0 &= \{([-8], [+8]), ([-9], [+9])\} \cup \\ &\quad \{[+9] XdB\# [-1] \$ [+1] \#a \mid a \in V_T \cup \{Y\}\} \cup \\ &\quad \{b\# [-2] \$ [+2] \#Y [-8] \mid b \in V_T \cup \{[+1]\}\} \\ R_k &= \{u\# [-3] \$ [+3] \#\alpha_k Y [-6] \mid u \in W \cup \{X\}\} \cup \\ &\quad \{[+5] X\# [-11] \$ [+11] \#v \mid v \in W\} \cup \\ &\quad \{([-3], [+3]), ([-11], [+11])\} \text{ for } 1 \leq k \leq n \end{aligned}$$

The remaining technical details of the proof are left to the reader. \square

The complexity of the GhP-system of type CR constructed in the preceding proof depends on the number of productions (and symbols) in the underlying grammar, which may give rise to a non-collapsing hierarchy with respect to the number of inner membranes. In contrast, for GhP-systems of type H we obtain an optimal result with respect to the membrane structure (compare with Lemma 3).

Theorem 5. *Any recursively enumerable language L can be generated by a GhP-system γ of type H with a fixed number of markers in the simplest non-trivial membrane structure $[0[1]_1]_0$.*

Proof. The main proof idea is the same as in the preceding proof, yet we also take advantage of the idea used in some proofs in [15] how to check the correspondence of the symbols marking the left and the right end of an intermediate string.

For simulating the production $p_k : \alpha_k \rightarrow \beta_k$, $1 \leq k \leq n$, in compartment 1, we start with applying

$$u\#\alpha_k Y[-3]\$[+3]\#Y_k[-6], u \in W \cup \{X\}, \text{ where } W = V_N \cup V_T \cup \{B\}, \text{ and} \\ [+7]X_k\beta_k\#[-2]\$[+2]X\#v, v \in W \cup \{Y_k\},$$

using the axioms $[+3]Y_k[-6]$ and $[+7]X_k\beta_k[-2]$.

In compartment 0, where we start with the axiom $[+2]XBSY[-3]$, the indices of the variables X and Y are decremented by applying

$$[+4]X_{i-1}\#[-7]\$[+7]X_i\#v, v \in W \cup \{d\}, 1 \leq i \leq n, \text{ and} \\ u\#Y_j[-6]\$[+6]\#Y_{j-1}[-5], u \in W, 1 \leq j \leq n,$$

using the axioms $[+4]X_{i-1}[-7]$ and $[+6]Y_{j-1}[-5]$.

Objects of the form $[+4]X_iwY_j[-5]$ can pass into compartment 1, where we apply

$$u\#Y_j[-5]\$[+5]\#Y_j[-6], u \in W, \text{ and} \\ [+7]X_i\#[-5]\$[+5]X_i\#v, v \in W \cup \{d\},$$

using the axioms $[+5]Y_j[-6]$ and $[+7]X_i[-5]$, for $0 \leq i, j \leq n-1$, in order to obtain $[+7]X_iwY_j[-6]$.

If in the first steps in compartment 1 the correctly matching splicing rules have been applied, we finally reach compartment 0 with an object of the form $[+7]X_0wY_0[-6]$, which is the only case that allows us to regain an object of the form $[+2]XwY[-3]$ by applying the splicing rules

$$[+2]X\#[-7]\$[+7]X_0\#v, v \in W, \text{ and} \\ u\#Y_0[-6]\$[+6]\#Y[-3], u \in W,$$

using the axioms $[+2]X[-7]$ and $[+6]Y[-3]$.

From terminating objects of the form $[+7]X_0dBw'Y_0[-6]$ with $w' \in L$ we obtain the terminal objects $[+1]w'[-2]$ by applying the splicing rules

$$[+1]\#X[-7]\$[+7]X_0dB\#a, a \in V_T \cup \{Y_0\}, \text{ and} \\ b\#Y_0[-6]\$[+6]Y\#[-2], b \in V_T \cup \{[+1]\},$$

using the axioms $[+1]X[-7]$ and $[+6]Y[-2]$.

From the list of axioms and splicing rules in the compartments 0 and 1 as described above the formal description of the GhP-system γ of type H can easily be completed; observe that some of the axioms, i.e., $[+6]Y_j[-5]$ and $[+5]Y_j[-6]$, for $0 \leq j \leq n-1$, can travel between compartment 0 and compartment 1, but this does not violate the correct functioning of γ :

$$\begin{aligned}
\gamma &= (B, B_T, R, A, [0[1]_1]_0, (A_0 \cup R_0, A_1 \cup R_1)) \\
B &= \mathbf{Z}' V^* \mathbf{Z}' \\
\mathbf{Z}' &= \{[+k], [-k] \mid 1 \leq k \leq 7\} \\
V &= V_N \cup V_T \cup \{d, B, X\} = W \cup \{d, X\} \\
B_T &= \mathbf{Z}'' V_T^* \mathbf{Z}'' \\
\mathbf{Z}'' &= \{[+k], [-k] \mid 1 \leq k \leq 2\} \\
R &= R_0 \cup R_1 \\
A &= A_0 \cup A_1
\end{aligned}$$

The sets of axioms are:

$$\begin{aligned}
A_0 &= \{[+2] X B S Y [-3], [+2] X [-7], [+6] Y [-3], [+1] X [-7], [+6] Y [-2]\} \cup \\
&\quad \{[+4] X_k [-7], [+6] Y_k [-5] \mid 0 \leq k \leq n-1\} \\
A_1 &= \{[+3] Y_k [-6], [+7] X_k \beta_k [-2] \mid 1 \leq k \leq n\} \cup \\
&\quad \{[+5] Y_k [-6], [+7] X_k [-5] \mid 0 \leq k \leq n-1\}
\end{aligned}$$

The splicing rules are collected in R_0 and R_1 as follows:

$$\begin{aligned}
R_0 &= \{[+4] X_{i-1} \# [-7] \$ [+7] X_i \# v \mid v \in W \cup \{d\}, 1 \leq i \leq n\} \cup \\
&\quad \{u \# Y_j [-6] \$ [+6] \# Y_{j-1} [-5] \mid u \in W, 1 \leq j \leq n\} \cup \\
&\quad \{[+2] X \# [-7] \$ [+7] X_0 \# v \mid v \in W\} \cup \\
&\quad \{u \# Y_0 [-6] \$ [+6] \# Y [-3] \mid u \in W\} \cup \\
&\quad \{[+1] \# X [-7] \$ [+7] X_0 d B \# a \mid a \in V_T \cup \{Y_0\}\} \cup \\
&\quad \{b \# Y_0 [-6] \$ [+6] Y \# [-2] \mid b \in V_T \cup \{[+1]\}\} \\
R_1 &= \{u \# \alpha_k Y [-3] \$ [+3] \# Y_k [-6] \mid u \in W \cup \{X\}, 1 \leq k \leq n\} \cup \\
&\quad \{[+7] X_k \beta_k \# [-2] \$ [+2] X \# v \mid v \in W \cup \{Y_k\}, 1 \leq k \leq n\} \cup \\
&\quad \{u \# Y_k [-5] \$ [+5] \# Y_k [-6] \mid u \in W, 0 \leq k \leq n-1\} \cup \\
&\quad \{[+7] X_k \# [-5] \$ [+5] X_k \# v \mid v \in W \cup \{d\}, 0 \leq k \leq n-1\}
\end{aligned}$$

According to the definitions and explanations given above, the reader may easily verify that $L(\gamma) = L$, which completes the proof. \square

The variables X_k and Y_k used in the preceding proof can be encoded in the markers; hence, allowing an unbounded number of markers, the proof idea elaborated above can also be used for showing the following result, i.e., there is a trade-off between the number of membranes and the number of markers to be used in GhP-systems of type CR:

Corollary 6. *Any recursively enumerable language L can be generated by a GhP-system γ of type CR in the simplest non-trivial membrane structure $[0[1]_1]_0$.*

Obviously, we can encode arbitrary additional data u in the axiom, i.e., we can take $[+k] X B S u Y [-k-1]$ instead of $[+k] X B S Y [-k-1]$ in compartment 0; moreover, starting with these axioms, the grammars can be considered as computing devices, the terminal strings $[+1] w' [-2]$ representing the output w'

computed from the input u . Hence, from the preceding theorems we also obtain the following result:

Corollary 7. *Any partial recursive function can be computed by a GhP-system of type CR and by a GhP-system of type H , respectively, using a fixed number of markers only.*

The complexity of the GhP-systems of type CR and H , respectively, in Corollary 7 is the same as of the corresponding GhP-systems constructed in the proofs of Theorems 4 and 5, i.e. for GhP-systems of type H the result is optimal with respect to the membrane structure and a fixed number of markers.

5 Conclusion

As it was already pointed out in [10], the idea of membrane structures offers a nearly unlimited variety of variants. In this paper we considered homogeneous (static) membranic structures with splicing or cutting/recombination rules inside and with the uniform permeability of the membranes to objects with the absolute value of the sum of electrical charges being equal to 1. The formal definition of molecular systems would also allow us to consider other objects than strings, e.g., graphs and the corresponding cutting and recombination rules (see [7]). A thorough investigation of the generative power of such variants of GhP-systems and their complexity for solving special problems remains for future research.

Further interesting features for P-systems can be found, for example, in [3], where the communication of objects is controlled by the concentration of the objects, and in [11], where the communication through membranes depends on the electrical charges of the atomic objects and the membranes. In [12], Gheorghe Păun gives an overview on the actual bibliography of P-systems and discusses a list of interesting problems related with membrane computing.

We finish with some open problems arising from the results discussed in this paper:

- Consider the simple GhP-systems of type CR using only recombination rules especially as in Lemmas 2 and 3:
 1. We possibly may have the chance to obtain an infinite non-collapsing hierarchy (compare with problem “m” in [12]) with respect to n in the membrane structure $[_0[_1]_1 \dots [_n]_n]_0$. The family of languages $\{ww^r \mid w \in \Sigma^*\}$ (where w^r denotes the mirror image of w) may be a candidate to constitute such a hierarchy with respect to the number of symbols in Σ .
 2. For the simplest membrane structure $[_0]_0$ we may obtain an infinite non-collapsing hierarchy with respect to the number of markers $[+k], [-k]$.
- Can the complexity (the number of membranes) of the GhP-systems of type CR constructed in the proof of Theorem 4 be reduced without increasing the number of markers?
- Can the the number of markers of the GhP-systems of type H constructed in the proof of Theorem 5 be reduced?

Acknowledgements. We gratefully acknowledge all the fruitful and interesting discussions with Gheorghe Păun concerning his brilliant ideas for various models of P-systems.

References

1. G. Berry and G. Boudol, *The chemical abstract machine*, Theoretical Computer Science 96 (1992), pp. 217-248.
2. J. Dassow and Gh. Păun, *On the power of membrane computing*, JUCS 5 (2) (1999), pp. 33-49.
3. J. Dassow and Gh. Păun, *Concentration controlled P systems*, submitted, 1999.
4. R. Freund, *Generalized P-systems*, FCT'99, Iasi, Romania, September 1999.
5. R. Freund, *Generalized P-systems with splicing and cutting/recombination*, WFLA'99, Iasi, Romania, September 1999.
6. R. Freund and F. Freund, *Test tube systems: When two tubes are enough*, DLT'99, Aachen, Germany, July 1999.
7. R. Freund and F. Freund, *Cutting and recombination of graphs*, AFL'99, Hungary, August 1999.
8. R. Freund and F. Freund, *Generalized homogeneous P-systems*, Research Report TU Wien, Austria, 2000.
9. Gh. Păun, *Computing with membranes*, Journal of Computer and System Sciences, 61, 1 (2000), pp. 108-143; and TUCS Research Report No. 208 (Nov. 1998).
10. Gh. Păun, *Computing with membranes: an introduction*, Bulletin EATCS 67 (Febr. 1999), pp. 139-152.
11. Gh. Păun, *Computing with membranes - a variant: P systems with polarized membranes*, International Journal of Foundations of Computer Science, 11, 1 (2000), pp. 167-182; and Auckland University, CDMTCS Report No. 098, 1999.
12. Gh. Păun, *Computing with membranes (P systems): Twenty-six research topics*, Auckland University, CDMTCS Report No. 119, 2000.
13. Gh. Păun, G. Rozenberg, and A. Salomaa, *DNA Computing: New Computing Paradigms* (Springer-Verlag, Berlin, 1998).
14. Gh. Păun, G. Rozenberg, and A. Salomaa, *Membrane computing with external output*, Fundamenta Informaticae, 41, 3 (2000) pp. 259-266; and TUCS Research Report No. 218 (Dec. 1998).
15. Gh. Păun and T. Yokomori, *Membrane computing based on splicing*, Preliminary Proc. Fifth Intern. Meeting on DNA Based Computers (E. Winfree and D. Gifford, eds.), MIT, June 1999, pp. 213-227.
16. Gh. Păun and T. Yokomori, *Simulating H systems by P systems*, JUCS 6 (2) (2000), pp. 178-193.
17. I. Petre, *A normal form for P-systems*, Bulletin EATCS 67 (Febr. 1999), pp. 165-172.
18. D. Pixton, *Splicing in abstract families of languages*, Theoretical Computer Science 234 (2000), pp. 135-166.

Computationally Inspired Biotechnologies: Improved DNA Synthesis and Associative Search Using Error-Correcting Codes and Vector-Quantization*

John H. Reif** and Thomas H. LaBean

Department of Computer Science, Duke University

Abstract. The main theme of this paper is to take inspiration from methods used in computer science and related disciplines, and to apply these to develop improved biotechnology. In particular, our proposed improvements are made by adapting various information theoretic coding techniques which originate in computational and information processing disciplines, but which we re-tailor to work in the biotechnology context.

(a) We apply Error-Correcting Codes, developed to correct transmission errors in electronic media, to decrease (in certain contexts, optimally) error rates in optically-addressed DNA synthesis (e.g., of DNA chips).

(b) We apply Vector-Quantization (VQ) Coding techniques (which were previously used to cluster, quantize, and compress data such as speech and images) to improve I/O rates (in certain contexts, optimally) for transformation of electronic data to and from DNA with bounded error.

(c) We also apply VQ Coding techniques, some of which hierarchically cluster the data, to improve associative search in DNA databases by reducing the problem to that of exact affinity separation. These improvements in biotechnology appear to have some general applicability beyond biomolecular computing.

As a motivating example, this paper improves biotechnology methods to do associative search in DNA databases. Baum [B95] previously proposed the use of biotechnology affinity methods (DNA annealing) to do massively parallel associative search in large databases encoded as DNA strands, but many remaining issues were not developed. Using in part our improved biotechnology techniques based on Error-Correction and VQ Coding, we develop detailed procedures for the following tasks:

(i) The database may initially be in conventional (electronic, magnetic, or optical) media, rather than the form of DNA strands. For input and output (I/O) to and from conventional media, we apply DNA chip technology improved by Error-Correction and VQ Coding methods for error-correction and compression.

* A postscript version of this paper is at URL

<http://www.cs.duke.edu/~reif/paper/Error-Restore/Error-Restore.ps>.

** Surface address: Department of Computer Science, Duke University, Box 90129, Durham, NC 27708-0129. E-mail: reif@cs.duke.edu. Supported by Grants NSF/DARPA CCR-9725021, CCR-96-33567, NSF IRI- 9619647 and EIA-0086015, ARO contract DAAH-04-96-1-0448, and ONR contract N00014-99-1-0406.

(ii) The query may not be an exact match or even partial match with any data in the database, but since DNA annealing affinity methods work best for these cases, we apply various VQ Coding methods for refining the associative search to exact matches.

(iii) We also briefly discuss how to extend associative search queries in DNA databases to more sophisticated hybrid queries that include also Boolean formula conditionals with a bounded number of Boolean variables, by combining our methods for DNA associative search with known BMC methods for solving small size SAT problems. For example, these extended queries could be executed on natural DNA strands (e.g., from blood or other body tissues) which are appended with DNA words encoding binary information about each strand, and the appended information could consist of the social security number of the person whose DNA was sampled, cell type, the date, further medical data, etc.

1 Introduction

1.1 Recombinant DNA Technology

DNA as a Storage Media. Recall that DNA is a linear molecule composed of 4 types of nucleotides, and thus provides a base 4 data encoding. DNA is an appealing media for data storage due to the very large amounts of data that can be stored in compact volume. It vastly exceeds the storage capacities of conventional electronic, magnetic, or even optical media. A gram of DNA contains about 10^{21} DNA bases, or about 10^8 terabytes. Hence, a few tens of grams of DNA may have the potential of storing all the human-made data currently stored in the world. DNA is about 10^8 times more compact than other storage media currently being used is. Most recombinant DNA techniques can be applied at concentrations of about 5 grams of DNA per liter of water.

Recombinant DNA Technology. Biotechnological methods, which are collectively known as recombinant DNA technology, have been developed for a wide class of operations on DNA strands. These operations include site-specific edits and splicing operations. In the DNA annealing operation, two single strands of DNA (with opposite 3' to 5' orientation) combine into a doubly stranded DNA if the DNA bases of these sequence are complementary (or nearly complementary) to each other. DNA separation techniques [KG97] make use of annealing [HG97] to separate out DNA strands that contain particular subsequences; typically one set of DNA strands is surface attached (e.g. to streptavidin-coated paramagnetic beads) and the affinity separation operates on another set of single stranded DNA which anneal to the affixed DNA. PCR [B94, R94] is a recombinant DNA operation that uses DNA annealing to amplifying the frequency of those DNA strands that have a particular chosen sequence.

1.2 Computationally Inspired Biotechnologies

Adleman has suggested the term *Computationally Inspired Molecular Technology* to describe molecular methods that are inspired by computational methods. While many of the more general applications to Molecular Technology are outside the scope of this paper, it is nevertheless a visionary concept which seems destined to have major impact

This paper has a narrower focus to biotechnology. The major theme of this paper is that we may provide improvements to biotechnology using methodologies similar to those developed by computer scientists. A field we term *Computationally Inspired Biotechnology* encompasses biotechnology methods that are inspired by computational methods. For example, the inventor of PCR has stated he was inspired by the technique of recursive programming in his discovery of PCR, which operates by a recursive doubling of concentrations of selected DNA strands.

As we shall see, our improvements in biotechnology will be made by adapting error-resilient and optimum rate techniques, which originate in computational disciplines.

A Motivating Example: Massively Parallel I/O using DNA Chips. For example, let us consider the conversion of a database from conventional electronic media to a “wet” database of DNA strands in solution or on solid support. To do these transformations of the database and queries, we investigate the use of a promising new biotechnology; namely that of DNA chips [FRP+91, DDSPL+93, PSS+94, BKH96, CYH+96], which provide the ability of highly parallel input/output over 2D surfaces. By use of photosensitive DNA-on-a-chip technology, 2D optical input is converted to DNA strands encoding the input data. If the database was initially in conventional electronic form, it can easily be displayed at a very high rate as a series of images in 2D optical form, and in principle parallel arrays of DNA chips can be used to synthesize strands of DNA each encoding the database elements. Furthermore, DNA chips can also be used for 2D optical output: using hybridization at the sites with fluorescent labeled DNA, the output can be read as a 2D image. (See Figure 1.)

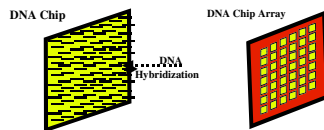


Fig. 1. DNA Chips and DNA Chip Arrays.

Each DNA chip can be optically addressed at up to 10^5 sites (and potentially many millions of sites in the immediate future), and each such chip is

small enough so that arrays of up to a few thousand chips can be placed on a 2D array compact enough so all chips can be addressed by a single optical system. Hence there is the potential of parallel synthesis of DNA at 10^8 sites or more (and potentially many billions of sites in the immediate future). Thus, this massively parallel DNA synthesis for input (and affinity detection of DNA strands for output) has a potential for achieving a rate of input/output to convention optical/electronic media in the order of gigabit rates or more. However, this all is without consideration of error rates, which are considerable, and represent the bulk of the technical challenge in this case. The most common errors in optically addressed synthesis of DNA are premature truncations of the growing strand and base deletions, although insertions and substitutions are also possible. (There are also a variety of other sources for further errors, such as differential sequence-dependant binding and secondary structure of the DNA strands.) The error rate in optically addressed DNA synthesis methods used for DNA chips is roughly 4% to 8% per base [MBD+97]. This corresponds to an expected error in every 12 to 25 base pairs.

The commercial chips such as Affymetrix utilize only a fraction of the 10^5 or so sites that are in principle optically addressable; the current maximum is about 42,000 sites, but the typical DNA chip uses only about 7,000 sites (see Table 1 of [LFGL99]). Due to the proprietary nature of the Affymetrix technology, it is not entirely clear what their synthesis error rates are for each type for possible error. For the technology at its current scale of just a few tens of thousands of sites, and with most strands under 20 base pairs, the synthesis error rate is not likely to be the most dominant limiting factor, and it is only one of a number of other factors that can impact the technology at the current scale. Nevertheless, methods for decreasing error rates due to optically addressed base synthesis might well impact future scalability of DNA chip technology by facilitating increased numbers of addressable sites and increased strand length.

1.3 Applications to Biomolecular Computing

Biomolecular Computing (BMC) makes use of biotechnology to do computation. Given the extreme compactness of DNA and the ability via recombinant DNA technology to execute operations on vast numbers of DNA strands in massively parallel fashion, BMC has impressive potential for molecular-scale computation. We consider two of the greatest challenges for BMC:

(a) Error-Resilient, High Rate I/O. Although Error-resilient techniques for BMC have been developed in [BL95a, CW97, DMGFS98, DMRGF+97, and DHS97], a key issue we consider is the transformation of inputs, originally in conventional electronic media, into sequences of DNA. Assuming the inputs are static, the conversion needs only to be done one time, but this still may be a nontrivial task. The application of DNA chips for I/O in Biomolecular Computing may be limited by their relatively high error rates. In this use of DNA

chips, each of the DNA strands synthesized may be quite long (likely well over 25 bases per strand), so as to transmit significant amounts of information, and then the majority of such strands can be expected to have at least one synthesis error. To address the synthesis error rate, we will apply Error-Correcting Coding methods.

(b) Scalable BMC Applications. Another key near term challenge [R96] in the field of BMC is to find applications of this technology that have the possibility of commercial utility in the near term, say in the next five years, and furthermore their resource requirements (number of recombinant DNA steps, volume of test tubes, etc.) should scale well so that future large scale demonstrations will be feasible. To date, there have been a number of BMC demonstrations and proposals for the solution of small size combinatorial search problems using separation techniques [A94, L95, ARRW96, BDL95, RWBCG+96] and surfaced based chemistry [CRFCC+96, LGCCL+96, BCGT96, CCCFF+97, LTCSC97, LFW+98, WQF+98]. But these applications are not scalable, since the volume grows exponentially with the problem size. Recently proposed applications of BMC that appear to be scalable include methods for hiding DNA [CRB99] and for encrypting DNA [GLR99], doing neural network learning [MYP98], and possibly certain other massively parallel computations [R95, GR98a], but their commercial utility may not be in the immediate future.

DNA Associative Search: A Scalable BMC Application. This paper takes associative search as our motivating example of a scalable BMC application, and provides solutions to the challenges listed above.

1.4 Organization of This Paper

In Section 1 we introduced recombinant DNA technology and the concept of computationally inspired biotechnologies In Sections 2 and 3 we present some coding methods used in computer science, namely Error-Correction coding and also Vector Quantization (VQ), that can be applied to improve biotechnology methods for error-resilient I/O and also optimum rate I/O between DNA databases and conventional media. In Section 4 we introduce our motivating example application: associative search and describe how VQ can be used to improve DNA associative search by to refining the associative search to exact matches. In Section 5 we briefly present methods for extending associative search queries to sophisticated hybrid queries that include also Boolean formula conditionals. In Section 6 we conclude the paper.

2 Error-Correction Methods from CS Adapted to Biotechnology

This section proposes experiment methods for repairing faulty oligonucleotides contained within surface-bound probe arrays. We propose the use of Error-

Correction DNA strands specifically designed to bind both error-containing and error-free probes. (Error-Correction strands can also act as templates for primer extension reactions, which will append error-free code words onto the 5' ends of all probes on the chip.) Our Error-Correction strands are composed of two segments: an error-containing portion (the suffix) designed to be complementary to the immobilized probe sequences and biased in its synthesis to contain similar types and quantities of errors as the faulty-probe sequence; and an error-free portion (the prefix) which will provide a error-free probe (or can be used as a template for primer extension to create a error-free probe). After hybridization with the Error-correction strands, the resulting duplex probes contain single-stranded overhangs with the error-free portion. (There is also evidence [BSS+94] that duplex probes containing single-stranded overhangs are substantially less error-prone than simple single-stranded probes, due to stacking interactions that provide increased stringency.) The design of these Error-Correction DNA strands is provided by information theoretic error-correction methods.

2.1 Known Error-Correction Methods

We will take inspiration from information theoretic error-correction methods (see Shannon [S48,S49], Hamming [H50], Berlekamp [B68], Pless [P82], Lint [L71]) used in computer science, with the goal of developing similar methods for various biotechnology applications.

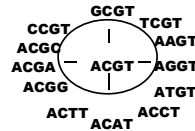
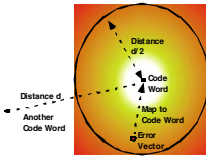


Fig. 2. Error-Correction by Mapping to **Fig. 3.** DNA sequences of generalized Hamming distance 1 from ACGT.

These error-correction methods make use of a set C of code words. When code words are altered by errors (during transmission or storage), they can be mapped back into the original set of error-free code words by an *Error-Correction* procedure. (See Figure 2.) In one of the simplest of these coding schemes, to encode each N -vector X whose elements range over $0, 1, \dots, b-1$, we use a code word $C(X)$ consisting of a N' -vector whose elements range over the same domain. In general, N' is larger than N ; that is we are *increasing the length of the encoding to gain error-resiliency*. The distance metric is defined so that the *generalized Hamming distance* between two vectors is the number of elements

where they differ. The code words are chosen so that there is at least distance $d = N' - N$ between each pair of code words. (See Figure 3.)

The *Maximum Likelihood Error-Correction* procedure simply maps each N' -vector to a code word that is closest in distance. Note that a restoration error does not occur as long as a vector is perturbed by at most d' errors, where d' is the floor of (the largest integer less than or equal to) $d/2$. The restoration error probability r is the likelihood that this Error-Correction procedure results in an error.

In the Boolean case (where the elements of vectors are 0 or 1), the error model (known as a *binary symmetric channel*) makes the assumption that each bit has a uniform, independent probability p of being flipped. In this case, the Hamming code [H50] provides an asymptotically optimum restoration error probability for any choice of the parameters N , $d = N' - N$, and p . In the more general case considered here, where the vector elements range over $0, 1, \dots, b - 1$, the error model provides an independent probability p of any element x of a vector being replaced by an element of $0, 1, \dots, b - 1 - x$ (note that the choice of the replacement element due to an error does not concern us, since that does not change the generalized Hamming distance).

There are known designs for sets of code words (e.g., see the Reed-Muller and BCH codes found in the texts [P82, L71]) which provide an asymptotically optimum restoration error probability $r = r(N, d, p, b)$ for any choice of the parameters N , d , p , and b . These codes have the property that, for a fixed p , there is a constant $c > 0$ such that the restoration error probability r can be made an inverse exponential function 2^{-cN} of N , and simultaneously, $d = N' - N$ can be made to asymptotically approach 0 as N grows¹. Our goal now is to take inspiration from this error-correction techniques and apply the concepts to improved biotechnology.

2.2 Applying Error-Correction Methods to Biotechnology

Recall that a *multiset* is a collection of items with possible repeats. Let the redundancy of a multiset S be the minimum number of repeats of any element of S . In the following, we assume multisets of DNA strands with very large redundancy, say at least 10^3 . Let us consider the case where we attempt chemical synthesis of a multiset S of single stranded DNA strands (the strands might be on a DNA chip or on other solid support), where each of the strands has the same total number of bases. We assume that the chemical synthesis has errors. Let us

¹ Similar results also hold for much more general channel error models, including discrete memoryless stationary processes. There are also more sophisticated coding methods known, for more complex models with non-uniform errors and also non-independent errors. For example, “burst” errors correspond to correlations of errors among consecutive elements, which necessitate modified methods for codings and the Error-Correction process.

suppose each base is synthesized within each of these strands with a uniform, independent probability p of a deletion error. We will approximately model these base deletions by replacements with other distinct bases. (To justify this, note that in the annealing process between two near-complementary strands s, s' , a base deletion in strand s generally would result in a local DNA base bulge in strand s' . (See again Figure 3. Short stretches of double-stranded DNA are depicted showing: a) exact Watson-Crick(WC) complementary matching; b) a mismatch (T-T) imbedded within a WC match region; and c) a WC match region surrounding a bulged base (T). The bulged base can be described as a deletion from the left-hand strand or an insertion into the right-hand strand.) The effect of this base bulge requires a very complicated energetic model; for simplicity, we approximate the effect of this base bulge to first order by a same length sequence of base mismatches, although these are not strictly energetically equivalent.)

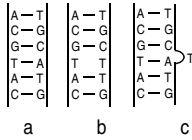


Fig. 4. Exact and Inexact Hybridization.

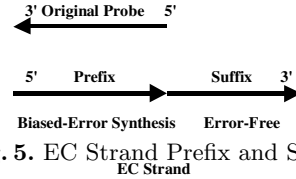


Fig. 5. EC Strand Prefix and Suffix.

Our Model for Synthesis Errors. So these synthesis errors with independent base deletions will be *approximated by an error model with a uniform, independent probability p of base replacement (without deletion error)*. (See Figure 4, which gives a 2D Projection of a local region in sequence space. Neighboring sequences are shown for a central tetramer (ACGT) with substitutions in the first position to the north, second to the east, third position south, and fourth west. Truncations, deletions and insertions are not shown.) This coincides with the uniform, independent replacement error model described in the above Subsection 2.1. Let $ERROR_p(S)$ be the multiset resulting from the attempted synthesis (with very high redundancy) of each of the single stranded DNA in multiset S with a uniform, independent probability p of base error. Now our challenge is to perform a Error-Correction procedure on the strands by purely biotechnology (i.e., recombinant DNA operations) means. In context of our problem, the coding theory parameters N, d, p , and b are set so:

- N is the number of bases in each of the strands of S ,
- $b = 4$ is the element domain size, since there are 4 DNA bases,
- p is given by the error model, and
- d is the minimum distance between each of the code words, and is a parameter that can be set to adjust the restoration error probability.

We choose a known design [P82, L71] for a set C of code words which provides an asymptotically optimum restoration error probability $r = r(N, d, p, 4)$ for choice of the parameters N, d, p . Our goal is to synthesize a multiset S of single stranded DNA, where each strand has N bases. For each strand s in S , we will define a *code word strand* $C(s)$ consisting of a single stranded DNA which gives the code word for the single stranded DNA s . Let $C(S)$ be the multiset of strands which are code word strands of the strands in S . We assume the encoding provides a restoration error probability $r = r(N, d, p, 4)$ for choice of the parameters N, d, p . There are two possible cases we consider:

(a) If S has the property that for each pair of strands in S differ by at least d bases (for example, we might expect this to be the case when S consists of a set of DNA words used in a DNA computation, and these DNA words were chosen so that all distinct pairs of these DNA words have low hybridization affinities), then we choose our code words $C(S)$ to be a set of DNA words with a 1 to 1 mapping C from S to $C(S)$, such that the elements of $C(S)$ all have length $N' = N$, and each s in S has low annealing affinity with the complement of $C(s)$ (note that this latter requirement implies that we can not set $C(s) = s$).

(b) Otherwise, we choose a known asymptotically optimum design [P82, L71] for the set C of code words, where for each s in S , code word strand $C(s)$ has $N' = N + d$ bases and moreover, s has low annealing affinity with the complement of $C(s)$. Since N' is larger than N , we are in this case *increasing the length of the encoding to gain error-resiliency*.

In either case, we now execute the following steps:

[0] **Initialization.** We first construct separately (by methods described in Subsection 2.3) a multiset of single stranded DNA strands, which we will call *Error-Correction (EC) strands*. (See Figure 5.) Each EC strand consists (in the 5' to 3' direction) of the prefix portion of the strand followed by the suffix portion of the strand. The prefix portion of the EC strand will be the result of synthesis of the complement of code word $C(s)$, with a uniform, independent probability p of DNA base synthesis errors. (Note that the synthesis of the prefix portion of the EC strand has the same error model as the synthesis of the code word $C(s)$.) The suffix portion of each EC strand will be the result $NEW(s)$ of synthesis of s with a much lower error rate: with a probability q (where $q \ll p$) of even a single DNA base synthesis error on that suffix.

[1] Rather than directly attempt chemical synthesis of multiset S , where each strand has N bases, we instead do chemical synthesis with this error model of the multiset $C(S)$ code word strands, resulting in a synthesized multiset $ERROR_p(C(S))$. Without loss of generality, we assume that these synthesized strands of $C(S)$ run from their attached 3' end to their unattached 5' end (however, note that we use the standard convention in our figures with arrows directed on the strands from the 5' end to the 3' end).

[2] Then we combine the EC strands with the DNA strands synthesized with this error model, and allow for hybridization. The hybridization products include doubly stranded DNA complexes with elements of $ERROR_p(S)$ hybridized with the corresponding prefix of the EC strands, with single stranded overhangs consisting of the suffix portion of the EC strands. (See Figure 6.)

In summary, we begin with the error-prone synthesis of a code word $C(s)$, and this results in a ssDNA overhang $NEW(s)$. Let S^* be the multiset of these single stranded overhangs $NEW(s)$, for each s in S . We say S^* approximates S with fidelity f if f lower bounds the probability that $NEW(s) = s$, for each s in S .

A precise statement of our result. We now show:

Theorem 1. If we employ a error-correction code with restoration error probability $r = r(N, d, p, 4)$, then S^* approximates S with fidelity $(1 - q)(1 - r)^2 \geq 1 - 2r - q$.

Proof: Suppose we attempt synthesis of a strand s in S in this error model. With likelihood $1 - r$, this yields a strand s' with at most d errors, where d is the floor of $(N' - N)/2$. Further suppose that the prefix portion of some EC strand is complementary to s' , and anneals to a strand s' . The likelihood that that EC strand was perturbed by less than d errors, also has likelihood $1 - r$, and conditional on this event, the further likelihood that the suffix $NEW(s)$ of that EC strand is s , is $1 - q$. Hence, the resulting single stranded overhang, consisting of the suffix portion $NEW(s)$ of that EC strand, is the strand s , with likelihood given by $(1 - q)(1 - r)^2$, which is at least $1 - 2r - q$. **QED**

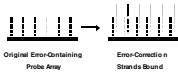


Fig. 6. Steps used to Error-Correct Synthesized DNA Strands, Resulting in Overhangs.

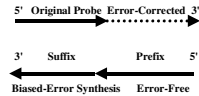


Fig. 7. An Alternative Method for Extension to Error-Free Probe Using Primer Extension on the EC' Strand.

Alternatively, we may instead wish to extend the original strands (say in the $5'$ to $3'$ direction) of S , so that the strand extending each of the original strands corresponds to its error-free codeword. In this case, we need to assume that these synthesized strands instead run from an attached $5'$ end to an unattached $3'$ end. We also need to slightly redefine the EC strands (see step [1']), and apply a well-known recombinant DNA operation known as primer-extension (e.g., see its previous use by [OGB97] for DNA computation). In this case, we define the code words so that the complement of s (rather than s , as in the case above) has low annealing affinity with the complement of $C(s)$.

We now execute the following modified steps (see Figure 8):

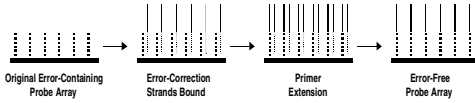


Fig. 8. Steps used to Error-Correct Synthesized DNA Strands, Resulting in Strand Extension.

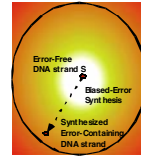


Fig. 9. Biased-Error DNA Synthesis.

[0'] We first construct separately (again, by methods described in Subsection 2.3) a multiset EC' of single stranded DNA strands, similar to the previously defined EC strands, except the suffix portion at the 5' end of the EC' strand is the complement of a possible result of synthesis of code word $C(s)$ (with the same synthesis error model), but the prefix portion at the 3' end of each EC strand is the *complement* to a strand s in S .

[1'] This step is the same as step [1] above.

[2'] This step is the same as step [2] above, except we anneal using the multiset EC' instead of multiset EC . This results in the hybridization products that include doubly stranded DNA complexes with elements of $ERROR_p(S)$ hybridized with the corresponding suffix of the EC strands, with single stranded overhangs consisting of the prefix portion of the EC' strands.

[3'] We then apply primer-extension. These single stranded overhangs provide templates used in the primer-extension procedure that provides extension of the strands of $ERROR_p(S)$ with the restored codewords, as intended.

Since this modified method is identical to the original one, except that complements are synthesized, the fidelity is the same as stated in Theorem 1, and in particular, the fidelity that these single stranded overhangs approximate multiset S is again lower bounded by $1 - 2r - q$.

2.3 Synthesizing the EC Strands

The remaining problem is to synthesize the EC strands as defined above in Section 2.2 (synthesis of the EC' strands is similar). This turns out to be the most interesting and technically demanding of the steps. (See Figure 9.) We consider a variety of methods:

(a) **Direct synthesis and purification of Error-Correction Strands.** Given detailed knowledge of the types and rates of sequence errors observed for oligonucleotides synthesized by light-directed chemistry on 2D chips, one could imagine constructing, by high fidelity solid-phase synthesis, a unique repair strand for each of the possible error sequences which then maps the error back to the intended probe sequence, as described above. This could be prohibitively expensive due to the large number of strands required. Nevertheless,

this could turn out to be the method of choice in the case of small-scale applications.

(b) Biased-Error Chemical Synthesis of Error-Correction Strands.

Here combinatorial synthesis is used for reducing the total number of separate syntheses required by simultaneously producing EC strands for a large number of possible flawed probes. Here the prefix portions of the EC strands are synthesized with errors in a manner mimicking the error model for the original probe strands, but where the suffix portion of the EC strands are synthesized with the lowest possible error rate. We propose the use of high-sequence-fidelity oligonucleotides produced by automated synthesis for the repair of errors incorporated during the production of probes by light-directed synthesis of addressable probe arrays. We will go from an expected error rate of 4–8% to something less than 1%. The proposed method would require a total number of syntheses and purifications equal to the intended diversity of the original probe array; while this may seem a daunting task, the payoff in error reduction and the corresponding increase in chip efficiency may be required for accurate I/O on critical computational or associative search applications.

Current protocols for chemical synthesis of oligonucleotides have been optimized to minimize sequence errors. The coupling efficiency for light-directed synthesis of DNA on planar solids are various estimated to be between 92% and 96% [MBD97, MH98] per cycle, compared to 98–99% for conventional synthesis using acid cleavage of methoxytrityl protecting groups on the same planar supports [MH98]. Higher coupling yields (> 99%) are routinely observed for automated, solid-phase oligonucleotide synthesis. The most commonly used method is the phosphite triester (phosphoramidite) procedure as modified by Beaucage and Caruthers [BC81]. The 3'-hydroxyl group of the first nucleotide is immobilized on a solid support (either controlled pore glass or a polystyrene-copolymer). The chain grows by the nucleophilic attack of the 5'-hydroxyl of the immobilized oligo on an activated 3'-phosphoramidite moiety of a 5'-protected building block. Reactive chemical groups on the bases are protected with various organic groups which are removed by treatment with ammonium hydroxide following the final coupling step. The reactive phosphates of the DNA phosphodiester backbone are protected throughout synthesis with -cyanoethyl which is also removed in the final ammonia deprotection step.

A quick review of the steps involved in chemical DNA synthesis will assist our discussion. Each cycle of chain elongation requires four steps: 1) deprotection, 2) coupling, 3) capping, 4) oxidation. *Deprotection* involves the removal of the dimethoxytrityl chemical-protecting group from the 5'-hydroxyl of the previous nucleotide. Trichloroacetic acid (1–3% w/v) is used in dichloromethane; the reaction requires less than one minute. *Coupling* provides for addition of the next nucleotide to the growing polymer. Excess soluble protected nucleosides and coupling reagent drive the reaction nearly to completion. Phosphoramidites

can not react directly with a free 5'-hydroxyl and must first be activated by a weak base like tetrazole. Tetrazole protonates the dialkylamino group of the phosphoramidite moiety and become a nucleophile, generating a very reactive tetrazolophosphane intermediate. Coupling reaction with these compounds is very fast (less than 2 minutes) and nearly quantitative. *Capping* is used to eliminate free 5'-hydroxyls which failed to react in the coupling step, thus capping decreases the frequency of sequence deletion products in the final oligo, converting them instead into truncations. Products of the capping reaction are terminated waste products and are no longer active during the remainder of synthesis. Acetic anhydride/N-methylimidazole is generally used as the capping reagent. *Oxidation* of the newly formed phosphite internucleotide linkage is unstable and must be oxidized to the more stable phosphate before chain extension can proceed. Iodine in tetrahydrofuran is a mild oxidizer with water as the oxygen donor. Reaction occurs within ≈ 30 seconds in very high yield.

Synthesis errors can occur at each of the steps in each cycle of the synthesis. The rates of different varieties of errors can be tuned in order to prepare oligos complementary to the probe libraries and containing, as an ensemble average, errors which match those spontaneously created in the light-directed chip synthesis. In order to achieve this tuning, very good data will be needed as to the types and frequencies of errors within the immobilized probes.

- **Tuning the rate of truncation errors:** Truncations occur when the deprotection step succeeds, the coupling step fails, and the capping step succeeds. Truncations on the chip can be converted into deletions if the capping step is eliminated. Truncations in the EC strands can be increased to match the chip rate by decreasing the efficiency of the coupling reaction (decreasing reaction time or by decreasing the concentrations of either the phosphoramidite or the activator (tetrazole/water). No changes are needed in the other steps.

- **Tuning the rate of deletion errors:** Deletion errors can occur if the deprotection step fails, or when it succeeds but the coupling and capping steps simultaneously fail. The easiest way to increase the rate of deletions on the EC strands is to decrease the efficiency of the deprotection step, either by decreasing the acid concentration or the reaction time or possibly both together.

- **Tuning the rate of substitution errors:** Substitution errors can be most easily incorporated into EC strands by spiking the nucleoside phosphoramidite stock solutions with an appropriate amount of contaminating phosphoramidite from the other nucleosides. For example, if a substitution rate of 1% observed for sites meant to be base G, then the G-monomer can be deliberately contaminated with 1% A-monomer. Reactivities of the phosphoramidites vary slightly with base identity, but at this stage, these differences will be ignored. We must also note that we will be synthesizing the complement of the error-containing strands and so must spike with T-monomer when we expect to basepair with erroneous A, etc.

(c) Other Methods for Generating Diverse Prefixes for EC Strands.

The relative simplicity of the biased-error chemical synthesis approach makes it the most appealing of methods for generating diverse prefixes for EC strands. However, for purposes of completeness, it should also be mentioned that there are a number of other possible methods.

Mutagenesis via Polymerase Enzymes. Sequence diversity can be generated during enzymatic polymerization of DNA by the use of polymerase enzymes and conditions which favor the introduction of mutations to the newly synthesized strands. Error rates of DNA polymerase enzymes vary from about one per million bases for high fidelity, proof-reading enzyme, to nearly one per hundred bases for highly mutagenic polymerization. In general the nature of errors may differ between chemical and enzymatic DNA synthesis, so biased-error chemical synthesis would most likely provide EC strands which most closely match the error profiles for probe DNA synthesized on a chip. Other mutagenesis Methods used for more general computations are given in [KG98].

DNA Self Assembly. It is also, in principle, possible to construct EC strands using a self-assembly method [R97, WLW+98, WYS96, LYK+00, LWR99, RLS00, MLR+00] involving a universal base pairing nucleotide like inosine to generate diverse populations of prefixes.

3 Adapting to Biotechnology VQ Methods Used in Computer Science

3.1 VQ Coding Methods Used in Computer Science

We next consider information theoretic Vector Quantization (VQ) Coding methods (see Gray [G90], Gersho, Gallager, and Gray [GGG91]) used in computer science for compressing data (such as speech and images) within bounded error. Again let $V = B^n$ be the set of all possible n -vectors over domain B of consecutive integers, and consider a database of vectors in V . VQ methods (which are also known as source coding and clustering methods) partition the vectors of the database into clusters of vectors, where each cluster is a subset of the database vectors. For each cluster G , the *center vector*, which may not be originally in the database of vectors, is the average of all the vectors of the cluster. The *radius* of a cluster is the maximum distance between any vector of the cluster to the center vector. There are well-known algorithms (Jain, Dubes [JD88]) which cluster the vectors of the database so the cluster radius is minimized and the average number of vectors in each cluster is bounded by a *cluster size* parameter m .

3.2 Applying VQ Coding Methods to Increase DNA Chip I/O

The clusters are enumerated and each assigned a *cluster index*, which is an integer that uniquely identifies the cluster. The number of clusters is a multiple

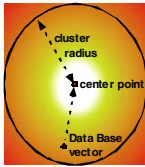


Fig. 10. A Vector Quantization Cluster.

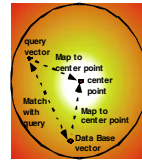


Fig. 11. Mapping the Query Vector to the Center of a Nearby Cluster.

$1/m$ of the original number of vectors of the database. Within a given cluster, each vector is approximated by the center point of the cluster and the code for a vector is an index to the cluster the vector is in. (See Figures 10 and 11.) Software for this process of VQ clustering has been developed by Eve Riskin's group at the University of Washington and can be obtained from their FTP site (<http://isdl.ee.washington.edu/compression/code/>).

Note that in contrast to Error-Correcting codes, the VQ coding induces errors, which are bounded by the choice of the clusters and can be tuned by setting the parameter m . For certain statistical source models for the data, for example memoryless or finite-state stationary processes (see Gray [G90], p 44), the resulting data-rate/distortion of VQ coding has been shown to be asymptotically optimal. However, natural data sources such as speech, images, and natural DNA [GT94, LY97, NW99] are known to be relatively uncompressible if it is compressed without errors e.g., via LZ compression [CT 91]), and so the performance of VQ coding on these data sources must be judged by empirical testing rather than by precise formulas. Extensive empirical testing of VQ coding (see Gray [G90], Gersho, Gallager, and Gray [GGG91]), has shown it to provide high factors of compression for many types of natural data, for example approximately 20 for speech and 30 for images, without much noticeable degradation.

An immediate application of VQ data clustering techniques is to improve I/O rates for transformation of electronic data to and from DNA with bounded error. Recall each vector of the database assumed to have a unique identification tag. After determining the clusters, their center points need to be transmitted (at $1/m$ the cost of transmitting the entire set of the database), and each vector v of the database is simply represented by a strand consisting of a series of DNA words encoding the unique identification tag for v and also an identification tag for the center point of the cluster that contains v . While for arbitrary databases, the performance (improved I/O rate) of VQ coding can not be precisely predicted by analytic techniques, we have noted that empirical evidence indicates it has excellent performance if the data base consists of speech or image data. The next

section discusses how to apply VQ coding methods to refine DNA associative search to exact matches.

4 Application to Associative Search

4.1 Definition of Associative Search

Let $V = B^n$ be the set of all possible n -vectors, whose elements range over a set B of consecutive numbers. Given two vectors u, v in V , let $\text{distance}(u, v) = |u_1 - v_1| + |u_2 - v_2| + \dots + |u_n - v_n|$, that is, sum of absolute values of the differences between corresponding elements of the vectors. We will use this distance metric in the context of associative search. The associative search problem assumes a database which is an ordered list of elements of V . In general, the input to an associative search query consists of a query vector in V and a distance bound d . The task is to search the entire database for those vectors (called the distance d near-matches) of the database that are of distance at most d from the query vector. If the distance bound is not specified, then the task is to find a vector (called the closest match) of the database that is of smallest distance from the query vector. (See Figure 12.) Each of the vectors of the database is assumed to have a unique identifying index in the list comprising the database, so the output from vectors can be specified by their indices.

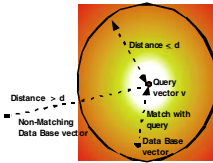


Fig. 12. Associative Search.

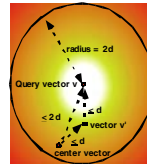


Fig. 13. Possibility Vectors of Distance $2d$ from the Query Vector.

Motivation. For example, the associative search problem arises as a basic task in the image-processing context. The image database is assumed to be preprocessed by a fixed procedure A to form an preprocessed database, which has a list of low level image attributes for each image or sub-image (e.g., $A(I)$ may apply a series of linear filters or perhaps a wavelet transform to the Image I). Given an input image I , we use A to determine the vector $A(I)$. Then an associative search in the preprocessed database provides the closest match to $A(I)$. This provides an index to that image in the image database whose attributes best match that

of the input image I . This motivates the development of an ultra-compact storage media (e.g., DNA) that supports highly parallel associative searches within the entire media.

Associative Search in Conventional Storage Media. In conventional highly compact storage (e.g., RAM, magnetic or optical), the time for an associative search though the entire database grows linearly with the size of the database. For example, the time for a sequential associative search through an entire database of 1 terabyte stored on optical disk may take at least a few hours. These searches can be sped up by a factor of P if P independent storage systems are accessed in parallel, but in conventional systems this degree of parallelism P is at most a few hundred. Image databases of size nearly 1000 terabytes (a terabyte = 10^{12} bytes) are being constructed by NASA and other federal agencies for space science and reconnaissance. The time for an associative search through a database of such size might, even with this degree of parallelism, could take at least a few days or even weeks.

4.2 DNA Annealing as an Associative Search Engine

An application previously proposed by Baum was massively parallel associative search in large databases, who sketched some approaches using known recombinant DNA methods for DNA ligation affinity separation such as such the use of streptavidin-coated paramagnetic beads. PCR also provides a way of doing associative search, since it uses DNA annealing to amplifying the frequency of those DNA strands that have a particular chosen sequence. (PCR is currently used for searching within large biological databases; for example, to fingerprint human DNA.) In the context of the DNA databases we consider, the database might have size $n = 1000$ terabytes.

Scalability. This application appears to be scalable, since (i) before the maximum concentration is reached, the number of recombinant DNA operations required (using PCR) for the search is independent of the size of the database and the database can be stored without a volume expansion, and (ii) after the maximum concentration is reached, the number of recombinant DNA operations required for the search is at most linear in the size of the database and the volume scales linearly.

Use of DNA Word Design in Associative Search. *DNA Word Design* is the problem of designing a library of short n -mer oligonucleotide sequences (DNA words) for information storage. These DNA words encode finite alphabets of symbols by appropriately chosen sets of DNA oligonucleotide sequences. Ideally, a good word design will maximize binding specificity, and minimize cross-binding affinity (mismatching) and also minimize secondary structure. DNA word design is crucial to error control in BMC, so is well studied. Some of the basic techniques required by this application, such as DNA word design, have already been

developed in prior work². Each element of a vector of the database is encoded by a DNA word. Since the number of possible values of each element is $|B|$, we require a library of $|B|$ distinct DNA words for encoding the possible elements at a given position in the vector. To decrease associative match misalignments, database vectors can use a distinct DNA word library of $|B|$ distinct DNA words (with minimal cross-binding affinity), for each of the n positions in the vector. Each vector of the database is thus encoded by a length n sequence of these DNA words, followed by DNA word encoding an identifying index to that vector.

4.3 Major Challenges Remaining

Nevertheless, key aspects of the associative search application needed development, including development of methods for the following key tasks (not considered by Baum):

(a) Input and Output (I/O) to Conventional Media: The database may initially be in conventional electronic media, rather than the form of DNA strands. The conversion of a static database needs only to be done one time, but then the queries also need to be so converted. Hence, the methods of Sections 2 and 3 can be used to improve I/O rates to and from conventional media, with error-resiliency and optimal I/O rate for a given error rate.

(b) Refining the Associative Search to Exact Affinity Separation: A key difficulty in the use of DNA annealing to do associative search is due to the high stringency of DNA annealing. Suppose the query vector v has a partial match with two data base vectors: (i) the vector v_1 matches the query nearly exactly except for a small number k of base mismatches scattered in the interior, while (ii) another vector v_2 matches the query exactly except for a much larger number $k' \gg k$ consecutive base mismatches at one end. Then v_1 is a much closer match with the query vector than v_2 . But if the vectors v_1, v_2 are encoded as DNA strands s_1, s_2 , respectively, and the query vector v is encoded in complementary fashion as a strand s , then it is quite possible that s might anneal to strand s_1 much better than to strand s_2 . The reason for this discrepancy is that in an annealing of two nearly complementary DNA strands, base mismatches that occur in scattered fashion in the interior of the strands can be less stable than mismatches at the end of strands. In general, in the annealing process between two single stranded DNA, the energetic properties of a set of mismatched bases in one strand (generally results in a local DNA bulge along the mismatch subsequence) varies dramatically depending on the position of the mismatches. Therefore, DNA annealing does not in general provide a very uniform metric for

² Researchers have provided DNA word designs using random strings [A94, L94], evolutionary search [DMRGF+97], error-correcting codes [W98], automated constraint-based procedures [HGL98], and other methods [A96, B96, DMGFS96, M96, GDNMF97, JK97a].

associative matching in the case of partial matches. Hence methods were needed for refining the associative search method to require only annealing on complementary sequences for which DNA annealing affinity methods work best, even if the query in not an exact match or even partial match with any data in the database.

4.4 Applying VQ Coding Methods to Associative Search: Refining the Associative Search to Exact Matches

DNA annealing affinity methods work best on complementary sequences. Yet, we need to process an associative match query, even if the query in not an exact match or even partial match with any data in the database. We now develop two methods for refining the associative search method to require only annealing on exactly complementary sequences. Both methods apply VQ-Coding clustering techniques.

(A) Associative Search with Given Match Distance. Our first method makes the assumption that we are given a bound d on the allowed match distance (recall that this is the distance between the query vector and the selected database vectors) for an associative match query. Using a conventional computer, we apply known VQ-Coding clustering techniques [G90, JD88], which provide a clustering of the database vectors so that the radius of each cluster is at most d . Recall that the elements of each vector in V range over a finite domain B . For each VQ cluster G of the database vectors, let $v(G)$ be the center point of G , and let the *possibility vectors* $P_{2d}(G)$ be the set of those vectors in V that are within distance $2d$ of the center point $v(G)$ of the cluster G (see Figure 13). We now describe our method for doing associative search:

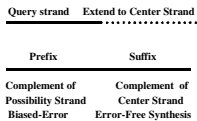


Fig. 14. Extension of the Query Strand to the Center Strand.

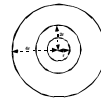


Fig. 15. Hierarchical Associative Search.

[0] Initialization: For each cluster G of the database vectors, we synthesize a DNA strand $s(G)$, to be called the *center stand* of that cluster G whose words encode the center point of the cluster, along with a unique identification tag. Then, using a biased-error chemical synthesis similar to the procedure given in Section 2.3 for synthesis of the EC strands, we construct from the center stand $s(G)$ of each of the cluster G , a multiset PS of single stranded DNA, each consisting of a prefix portion that encodes the complement of a possibility

vector in $P_{2d}(G)$, followed by a suffix portion consisting of the complement of the center stand $s(G)$. This step is done only once, for a static data base.

Associative Query Processing

Input: A query vector v .

[1] We synthesize a multiplicity of DNA strands, called *query strands* encoding the query vector v .

[2] We then combine these query strands with the multiset PS . The hybridization products include doubly stranded DNA complexes. Each of these consist of a query strand hybridized with a corresponding prefix of a PS strand, with single stranded overhang consisting of the suffix portion of the PS strand consisting of the complement of the center strand (see Figure 14).

[3] Primer extension is applied to these hybridization products with overhangs, so that each query strand is extended to include the center strand as its new suffix, forming a *result strand*.

[4] Then we denature to single stranded DNA, and separate out the result strands using affinity separation with strands complementary to the center strands.

[5] **Output:** Finally, we output to conventional media (e.g., by the use of DNA chips) the center vectors corresponding to the suffix portions of the result strands. We use a conventional computer to enumerate the vectors of the clusters of each of these center vectors and to determine which of each cluster's vectors are of distance at most d from the query vector v . (Recall we have already precomputed, using a conventional computer, the clustering and the center of each cluster. Hence these result strands suffice for us to determine the cluster index and list of vectors of the clusters.)

The main point is that this construction reduces the associative search problem to that of finding just exact matches (via complementary hybridization), and this can be done very effectively by known DNA annealing methods (e.g., PCR). Note that the set of possibility vectors $P_d(G_i)$ defines an n -dimensional ball, with respect to the defined distance metric, with radius $2d$ and centered at the center point of cluster G_i . Since elements of the vectors range over a set of size $b = |B|$, the size of the set $P_{2d}(G)$ of possibility vectors of any cluster G is at most $b^{2d} - 1$. For bounded, modest size b and d (where say $b = 28 = 64$ and $d = 5$), we are able to do this construction in the DNA domain, since DNA is quite compact.

Proof of the Algorithm and Analysis. We now show:

Theorem 2. The final output determined in step [5] consists of all database vectors that are at most distance d to the query vector. Furthermore, the number of distinct database vectors that are enumerated by a conventional computer in step [5] is at most $b^{2d} - 1$.

Proof: Let G_1, \dots, G_j be the selected clusters whose centers are of distance at most d from the query vector v , and let v_1, \dots, v_j be the centers of these clusters selected in step [4]. Note that the query vector, represented by a DNA

strand, will be included among the possibility vectors of each of those clusters. We need to prove that these center vectors are of distance at most $2d$ from the query vector, and their clusters include all database vectors that are at most distance d to the query vector. The elements of each cluster G_i are of distance at most d from their center vector v_i , and each center vector v_i is of distance at most d from the query vector v . Hence, the vectors in these clusters are at most twice this distance, that is of most distance $2d$, to the query vector v . So the number of distinct database vectors that are enumerated in step [5] is at most $b^{2d} - 1$. Furthermore, consider any database vector v' that is at most distance d to the query vector v . Then v' will be in a cluster G whose center $v(G)$ is at most distance d from v' , and so the center $v(G)$ will be of distance at most $2d$ from the query vector v . Hence, v' will be included as an element of one of these selected clusters G_1, \dots, G_j . **QED**

(B) Hierarchical Associative Search. Our next method, which we just sketch, is a more complex procedure which makes only the assumption that the match distance be upper bounded by d . For example, we will assume that the match distance be at most the radius of the entire set of database vectors. We apply known hierarchical VQ-Coding clustering techniques [G90, JD88]. These make a series of distinct clusterings of the data at exponentially increasing cluster radius. These hierarchical sequence of clusterings can be represented by a tree, whose root is the original set of vectors, and where each level of the tree is a clustering, refining the clustering at the previous level, and with cluster radius half that of the previous level (see Figure 15). Given an associative match query, let the *near query vectors* within radius r be the set $P_r(G)$ of possibility vectors of a hypothetical cluster of radius r with center vector being the input query vector. For each level of the tree with a radius bound of say r , we synthesize a multiset of DNA strands, which will be called *near query strands* which encode the near query vectors within radius r . To execute an associative search, we do not search just using the input query vector, and instead augment the query vector with these near query vectors of increasing radius. The search proceeds on decreasing levels $= 0, 1, \dots$ starting at the lowest level of the tree. At each of these levels, of radius bound of say r , we execute an affinity separation to determine if there is any exact matches between the near query vector within that radius r that exactly matches a vector of the database. We terminate where either r exceeds d , or at the first level where at least one near query vector exactly matches a vector of the database. Again, the main point of this construction is to reduce the associative search problem to that of finding just exact matches (via complementary hybridization), and again this can be done very effectively by known DNA annealing methods such as PCR. Note that the number possibility vectors of any cluster of radius r is at most $b^r - 1$, and this is upper bounded by $b^{2d} - 1$, as in the previous method. Hence for bounded, modest size b and d , we are able to do this construction in the ultra-compact DNA domain.

5 Extension of Associative Search to Include Boolean Conditionals

Finally, we briefly describe how to extend associative search queries to sophisticated hybrid queries that include also Boolean formula conditionals (with a small bounded number n' of Boolean variables), by combining our methods for DNA associative search with known BMC methods for solving the SAT problem (e.g., using surface chemistry techniques [CRFCC+96, LGCCL+96, BCGT96, CCCFF+97, LTCSC97, LFW+98, WQF+98]). We assume that each of the vectors of the database are augmented with “digital tag vectors” consisting of a list of n' Boolean values, encoding binary information about the vector. An extended query consist of (i) a query vector to be matched with and (ii) a Boolean formula to be satisfied. The extended query requires finding those database vectors that closely match the query vector and also whose Boolean variables satisfy the queries Boolean formula. The DNA strands encoding these database vectors also are augmented with prefix *digital tag strands* consisting of a sequence of DNA words encoding these Boolean values.

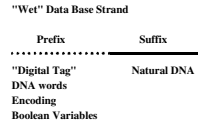


Fig. 16. A Natural DNA Strand with a Digital Tag Prefix.

For example, the extended database might consist of natural DNA strands (e.g., from blood or other body tissues) onto which are appended at their 5' ends prefix digital tag strands (see Figure 16) consisting of DNA words encoding identifying information about each strand (such as social security number of the person whose DNA was sampled, cell type, the date, further medical data, etc.). The digital tag strands may have been constructed by previous BMC processing. There is also proposed methods [LL97] for recoding natural DNA, by the use of nonstandard DNA bases, into a form more amenable to computation.

Our technique is to execute the extended query in two stages:

- (a) We first execute the Boolean formula portion of the query as a SAT problem, using biomolecular computing techniques previously developed (e.g., using surface chemistry techniques referenced above). These method can be implemented so that those strands not encoding SAT solutions are deleted, and all the remaining DNA strands satisfy the Boolean formula.
- (b) Then we execute the associative search, as previously described in Section 3, on the remaining strands, to find the closest match to the query vector that satisfies the query's Boolean formula.

6 Conclusion

We have provided some examples of how ideas from information processing disciplines can be applied to biotechnology. (As we have already pointed out, this is not the first time this was done, if we consider PCR as essentially a recursive algorithm for selective strand amplification.) We believe that this approach of “Computationally Inspired Biotechnologies” will be a profitable approach for overcoming key biotechnology challenges remaining, for example:

- increased affinity selectivity for a wider range of molecules, and
- decreased errors in chemical synthesis.

Moreover, in the immediate future, the separation between computational and biological technologies should narrow. As the miniaturization of biotechnology continues, we can expect DNA chip technology to have also MEMS microflow devices and computational processing capability as well (see Gehani and Reif [GR98] and Suyama [S98]).

References

- [A94] Adleman, L.M., “Molecular Computation of Solution to Combinatorial Problems”, *Science*, 266, 1021, (1994).
- [ARRW96] Adleman, L.M., P.W.K. Rothmund, S. Roweis, E. Winfree, “On Applying Molecular Computation To The Data Encryption Standard”, 2nd Annual DIMACS Meeting on DNA Based Computers, Princeton, June, 1996
- [BCGT96] Bach, E., A. Condon, E. Glaser, and C. Tanguay, “Improved Models and Algorithms for DNA Computation”, *Proc. 11th Annual IEEE Conference on Computational Complexity, J. Computer and System Sciences*, to appear.
- [B94] Barnes, W.M., “PCR amplification of up to 35-kb DNA with high fidelity and high yield from bacteriophage templates”, *Proc. Natl. Acad. Sci.*, 91, 2216–2220, (1994).
- [B95] Baum, E. B., “How to build an associative memory vastly larger than the brain”, *Science*, pp 583-585, April 28, 1995.
- [B96] Baum, E. B. “DNA Sequences Useful for Computation, 2nd Annual DIMACS Meeting on DNA Based Computers”, Princeton University, June 1996.
- [BC81] Beaucage, S.L., and Caruthers, M.H. (1981). “Deoxynucleoside phosphoramidites- A new class of key intermediates for deoxypolynucleotide synthesis”, *Tetrahedron Lett.* 22, 1859-1862.
- [B68] E. R. Berlekamp, “Algebraic Coding theory”, McGraw-Hill Book Company, NY (1968).
- [BKH96] Blanchard, A. P., R. J. Kaiser and L. E. Hood, “High-density oligonucleotide arrays”, *Biosens. Bioelec.*, Vol. 11, 687-690, (1996).
- [BDL95] Boneh, D., C. Dunworth, R. Lipton, “Breaking DES Using a Molecular Computer”, Princeton CS Tech-Report number CS-TR-489-95, (1995).

- [BL95a] Boneh, D., and R. Lipton, "Making DNA Computers Error Resistant", Princeton CS Tech-Report CS-TR-491-95, Also in 2nd Annual DIMACS Meeting on DNA Based Computers, Princeton University, June 1996.
- [BL95b] Boneh, D., and R. Lipton, "A Divide and conquer approach to DNA sequencing", Princeton University, 1996.
- [BSS+94] N. E. Broude, T. Sano, C. L. Smith, and C. R. Cantor, "Enhanced DNA Sequencing by hybridization", *Proc. Natl. Acad. Sci.*, Vol. 91, pp. 3071-3076, (April, 1994).
- [CCCCFF+97] Cai, W., A. Condon, R.M. Corn, Z. Fei, T. Frutos, E. Glaser, Z. Guo, M.G. Lagally, Q. Liu, L.M. Smith, and A. Thiel, "The Power of Surface-Based Computation", *Proc. First International Conference on Computational Molecular Biology (RECOMB97)*, January, 1997.
- [CRFCC+96] Cai, W., E. Rudkevich, Z. Fei, A. Condon, R. Corn, L.M. Smith, M.G. Lagally, "Influence of Surface Morphology in Surface-Based DNA Computing", Submitted to the 43rd AVS National Symposium, Abstract No. BI+MM-MoM10, (1996).
- [CYH+96] Chee, M., R. Yang, E. Hubbell, A. Berno, X. C. Huang, D. Stern, J. Winkler, D. J. Lockhart, M. S. Morris and S. P. A. Fodor, "Accessing genetic information with high-density DNA arrays", *Science*, Vol. 274, 610-614, (1996).
- [CW97] Chen, J., and D. Wood, "A New DNA Separation Technique with Low Error Rate", Third Annual DIMACS Workshop on DNA Based Computers, University of Pennsylvania, June 23-26, 1997. Published in *DNA Based Computers, III, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, Vol 48 (ed. H. Rubin), American Mathematical Society, (1999).
- [CRB99] Clelland, C.T., Risca, V., and C. Bancroft. "Genomic Steganography: Amplifiable Microdots". To appear in *Nature*, 1999.
- [CT 91] Cover, T. M. and J. A. Thomas, "Elements of Information Theory", John Wiley, New York, NY, (1991).
- [DMGFS96] Deaton, R., R.C. Murphy, M. Garzon, D.R. Franceschetti, and S.E. Stevens, Jr., "Good encodings for DNA-based solutions to combinatorial problems", *Proceedings of the 2nd Annual DIMACS Meeting on DNA Based Computers*, June 1996.
- [DMGFS98] Deaton, R., R.C. Murphy, M. Garzon, D.R. Franceschetti, and S.E. Stevens, Jr., "Reliability and efficiency of a DNA-based computation", *Phys. Rev. Lett.* 80, 417-420 (1998).
- [DMRGF+97] Deaton, R., R.C. Murphy, J.A. Rose, M. Garzon, D.R. Franceschetti, and S.E. Stevens, Jr., "A DNA Based Implementation of an Evolutionary Search for Good Encodings for DNA Computation", *ICEC'97 Special Session on DNA Based Computation*, Indiana, April, 1997.
- [DHS97] Deputat, M., G. Hajduczuk, E. Schmitt, "On Error-Correcting Structures Derived from DNA", Third Annual DIMACS Workshop on DNA Based Computers, University of Pennsylvania, June 23-26, 1997. Published in *DNA Based Computers, III, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, Vol 48 (ed. H. Rubin), American Mathematical Society, (1999).
- [DDSPL+ 93] Drmanac, R., S. Drmanac, Z. Strezoska, T. Paunesku, I. Labat, M. Zeremski, J. Snoddy, W. K. Funkhouser, B. Koop, L. Hood, and R. Crken-

- jakov "DNA Sequence Determination by Hybridize: A Strategy for Efficient Large-Scale Sequencing", *Science*, 260, 1649–1652, (1993).
- [FRP+91] Fodor, S. P. A., J. L. Read, C. Pirrung, L. Stryer, A. T. Lu and D. Solas, "Light-directed spatially addressable parallel chemical synthesis", *Science*, Vol. 251, 767–773, (1991).
- [FTCSC97] Frutos, A.G., A.J. Thiel, A.E. Condon, L.M. Smith, R.M. Corn, "DNA Computing at Surfaces: 4 Base Mismatch Word Design", Third Annual DIMACS Workshop on DNA Based Computers, University of Pennsylvania, June 23–26, 1997. Published in *DNA Based Computers, III, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, Vol 48 (ed. H. Rubin), American Mathematical Society, (1999).
- [GDNMF97] Garzon, M., R. Deaton, P. Neathery, R.C. Murphy, D.R. Franceschetti, S.E. Stevens Jr., "On the Encoding Problem for DNA Computing", Third Annual DIMACS Workshop on DNA Based Computers, University of Pennsylvania, June 23–26, 1997. Published in *DNA Based Computers, III, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, Vol 48 (ed. H. Rubin), American Mathematical Society, (1999).
- [GLR99] Gehani, A., T. H. LaBean, and J.H. Reif, "DNA-based Cryptography", 5th DIMACS Workshop on DNA Based Computers, MIT, June, 1999. *DNA Based Computers, V, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, (ed. E. Winfree), American Mathematical Society, 2000. <http://www.cs.duke.edu/~reif/paper/DNAcrypt/crypt.ps>
- [GR98] Gehani, A. and J. Reif, "Micro flow bio-molecular computation", 4th DIMACS Workshop on DNA Based Computers, University of Pennsylvania, June, 1998. *DNA Based Computers, IV, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, (ed. H. Rubin), American Mathematical Society, (1999). Also, special issue of *Biosystems*, Vol. 52, Nos. 1–3, (ed. By L. Kari, H. Rubin, and D. H. Wood), pp 197–216, (1999). <http://www.cs.duke.edu/~reif/paper/geha/microflow.ps>.
- [GGG91] Gersho, A., R. Gallager, and R. M. Gray, "Vector Quantization and Signal Compression", Kluwer Academic Publishers, (1991).
- [GFBCL+96] Gray, J. M. T. G. Frutos, A.M. Berman, A.E. Condon, M.G. Lagally, L.M. Smith, R.M. Corn, "Reducing Errors in DNA Computing by Appropriate Word Design", University of Wisconsin, Department of Chemistry, October 9, 1996.
- [G90] Gray, R. M., "Source Coding Theory", Kluwer Academic Publishers, Boston, (1990).
- [GT94] Grumbach, S., and F. Tahi, "Compression of DNA Sequences", *Proceedings of the IEEE Data Compression Conference (DCC'94)*, Snowbird, UT, 72–82, March 1994.
- [H50] Hamming, R. W., "Error Detection and error correcting codes", *Bell System Technical Journal*, Vol. 29, 147–160, (1950).
- [HGL98] Hartemink, A., David Gifford, J. Khodor, "Automated constraint-based nucleotide sequence selection for DNA computation", 4th DIMACS Workshop on DNA Based Computers, University of Pennsylvania, June, 1998. *DNA Based Computers, IV, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, (ed. H. Rubin), American Mathematical Society, (1999).
- [HG97] Hartemink, A.J., D.K. Gifford, "Thermodynamic Simulation of Deoxyoligonucleotide Hybridize for DNA Computation", Third Annual DIMACS

- Workshop on DNA Based Computers, University of Pennsylvania, June 23-26, 1997. Published in DNA Based Computers, III, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol 48 (ed. H. Rubin), American Mathematical Society, (1999).
- [JD88] Jain, A. K. and R. C. Dubes, "Algorithms for clustering data," Prentice Hall, Englewood Cliffs, N.J., (1988).
- [KG97] Khodor, J., and David K. Gifford, "The Efficiency of Sequence-Specific Separation of DNA Mixtures for Biological Computing", Third Annual DIMACS Workshop on DNA Based Computers, University of Pennsylvania, June 23-26, 1997. Published in DNA Based Computers, III, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol 48 (ed. H. Rubin), American Mathematical Society, (1999).
- [KG98] Khodor, J., D. Gifford, "Design and implementation of computational systems based on programmed mutagenesis", 4th DIMACS Workshop on DNA Based Computers, University of Pennsylvania, June, 1998. DNA Based Computers, IV, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, (ed. H. Rubin), American Mathematical Society, (1999).
- [FLGL99] Lipschutz, R.J., Fodor, P.A., Gingeras, T.R., and Lockhart, D.J. Nature Genetics Supplement, vol 21, pp 20-24 (1999).
- [LYK+00] LaBean, T. H., Yan, H., Kopatsch, J., Liu, F., Winfree, E., Reif, J.H. and Seeman, N.C., "The construction, analysis, ligation and self-assembly of DNA triple crossover complexes", J. Am. Chem. Soc. 122, 1848-1860 (2000). www.cs.duke.edu/~reif/paper/DNAtiling/tilings/JACS.pdf
- [LWR99] LaBean, T. H., E. Winfree, J. H. Reif, "Experimental Progress in Computation by Self-Assembly of DNA Tilings", 5th International Meeting on DNA Based Computers(DNA5), MIT, Cambridge, MA, (June, 1999). To appear in DIMACS Series in Discrete Mathematics and Theoretical Computer Science, ed. E. Winfree, to appear American Mathematical Society, 2000. <http://www.cs.duke.edu/~thl/tilings/labean.ps>
- [LL97] Landweber, L.F. and R. Lipton, "DNA 2 DNA Computations: A Potential 'Killer App'?", 3rd Annual DIMACS Meeting on DNA Based Computers, University of Pens., (June 1997).
- [L71] van Lint, J. H., "Coding Theory", Lecture Notes in Mathematics, Springer Verlag, NY, (1971).
- [L95] Lipton, R.J. "DNA Solution of Hard Computational Problems", Science, 268, 542-845, (1995).
- [LFW+98] Liu, Q., A. Frutos, L. Wang, A. Thiel, S. Gillmor, T. Strother, A. Condon, R. Corn, M. Lagally, L. Smith, "Progress towards demonstration of a surface based DNA computation: A one word approach to solve a model satisfiability problem", 4th DIMACS Workshop on DNA Based Computers, University of Pennsylvania, June, 1998. DNA Based Computers, IV, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, (ed. H. Rubin), American Mathematical Society, (1999).
- [LGCCL+96] Liu, Q., Z. Guo, A.E. Condon, R.M. Corn, M.G. Lagally, and L.M. Smith, "A Surface-Based Approach to DNA Computation", Proc. 2nd Annual Princeton Meeting on DNA-Based Computing, June 1996.
- [LTCSC97] Liu, Q., A.J. Thiel, A.G. Frutos, R.M. Corn, L.M. Smith, "Surface-Based DNA Computation: Hybridize and Destruction", Third Annual DIMACS Workshop on DNA Based Computers, University of Pennsylvania, June 23-26, 1997. Published in DNA Based Computers, III, DIMACS Series in

- Discrete Mathematics and Theoretical Computer Science, Vol 48 (ed. H. Rubin), American Mathematical Society, (1999).
- [LY97] Loewenstern, D. and Yainilos, P., "Significantly lower entropy estimates for natural DNA sequences", J.A Storer and M Cohn (Eds.), IEEE Data Compression Conference, Snowbird, UT, pp. 151-161, (March, 1997).
- [MLR+00] Mao, C., T.H. LaBean, J. H. Reif, and N.C. Seeman, "An Algorithmic Self-Assembly", *Nature*, Sept 28, (2000). www.cs.duke.edu/~reif/paper/SELF-ASSEMBLE/AlgorithmicAssembly.pdf
- [MH98] Marshall, A., Hodgson, J. 1998 *Nature Biotechnology* 16, pp 27-31.
- [MBD+97] McGall, G.H., Barone, A.D., Diggelmann, M., Ngo, N., Gentalen, E., and Fodor, S.P.A. "The Efficiency of Light-Directed Synthesis of DNA Arrays on Glass Substrates". *J. Am. Chem. Soc.*, 119(22): 5081-5090, (1997).
- [MYP98] Mills, A., B. Yurke, P. Platzman, "Error-tolerant massive DNA neural-network computation", 4th DIMACS Workshop on DNA Based Computers, University of Pennsylvania, June, 1998. DNA Based Computers, IV, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, (ed. H. Rubin), American Mathematical Society, (1999).
- [M96] Mir, K.U., "A Restricted Genetic Alphabet for DNA Computing", 2nd Annual DIMACS Meeting on DNA Based Computers, Princeton University, (June 1996).
- [NW99] Nevill-Manning, C.G. and I.H. Witten, "Protein is Incompressible", J.A Storer and M Cohn (Eds.), IEEE Data Compression Conference, Snowbird, UT, pp. 257-266, (March, 1999).
- [PSS+94] Pease, A. C. , D. Solas, E. J. Sullivan, M. T. Cronin, C. P. Holmes and S. P. Fodor, "Light-generated oligonucleotide arrays for rapid DNA sequence analysis", *Proc. Natl Acad. Sci. USA*, Vol. 91, 5022-5026, (1994).
- [P82] V. Pless, "Introduction to the theory of error-correcting codes," John Wiley and Sons, , NY (1982).
- [OGB97] Orlian, M., F. Guarnieri, C. Bancroft, "Parallel Primer Extension Horizontal Chain Reactions as a Paradigm of Parallel DNA-Based Computation", Third Annual DIMACS Workshop on DNA Based Computers, University of Pennsylvania, June 23-26, 1997. Published in DNA Based Computers, III, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol 48 (ed. H. Rubin), American Mathematical Society, (1999).
- [R93] Reif, J. (ed.), *Synthesis of Parallel Algorithms*, Morgan Kaufmann, (1993).
- [R95] Reif, J.H., "Parallel Molecular Computation: Models and Simulations", Seventh Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA95), ACM, Santa Barbara, 213-223, June 1995. *Algorithmica*, special issue on Computational Biology, 1999. (<http://www.cs.duke.edu/~reif/paper/paper.html>)
- [R97] Reif, J.H., "Local Parallel Biomolecular Computation", 3rd DIMACS Meeting on DNA Based Computers, Univ. of Penns., (June, 1997). DIMACS Series in Discrete Mathematics and Theoretical Computer Science, ed. H. Rubin, (1999). (<http://www.cs.duke.edu/~reif/paper/Assembly.ps> and [/Assembly.fig.ps](http://www.cs.duke.edu/~reif/paper/Assembly.fig.ps))
- [R98] Reif, J.H., "Paradigms for Biomolecular Computation", First International Conference on Unconventional Models of Computation, Auckland, New Zealand, January 1998. *Unconventional Models of Computation*, edited by C.S. Calude, J. Casti, and M.J. Dinneen, Springer Pub., Jan. 1998, pp 72-93. (<http://www.cs.duke.edu/~reif/paper/paradigm.ps>)

- [RLS00] J.H. Reif, T. H. LaBean, and Seeman, N.C., Challenges and Applications for Self-Assembled DNA Nanostructures, Invited paper, Sixth International Meeting on DNA Based Computers (DNA6), DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Leiden, The Netherlands, (June, 2000) ed. A. Condon. To be published by Springer-Verlag as a volume in Lecture Notes in Computer Science, (2000). <http://www.cs.duke.edu/~reif/paper/SELFASSEMBLE/selfassemble.ps>
- [R94] Roberts, S.S., "Turbocharged PCR", Jour. of N.I. H. Research, 6, 46-82, (1994).
- [RDGS97] Rose, J.A., R. Deaton, M. Garzon, and S.E. Stevens Jr., "The Effect of Uniform Melting Temperatures on the Efficiency of DNA Computing", Third Annual DIMACS Workshop on DNA Based Computers, University of Pennsylvania, June 23-26, 1997. Published in DNA Based Computers, III, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol 48 (ed. H. Rubin), American Mathematical Society, (1999).
- [RWBCG+96] Roweis, S., E. Winfree, R. Burgoyne, N.V. Chelyapov, M.F. Goodman, P.W.K. Rothmund, L. M. Adleman, "A Sticker Based Architecture for DNA Computation", 2nd Annual DIMACS Meeting on DNA Based Computers, Princeton University, June 1996, Also as Laboratory for Molecular Science, USC technical report A Sticker Based Model for DNA Computation, May 1996.
- [R96] R96 Rubin, H. "Looking for the DNA killer app.", Nature, 3, 656-658, (1996).
- [S48] Shannon, C. E., "A mathematical theory of communication", Bell System Technical Journal, Vol. 27, 379-423 and p 623-656, (1948).
- [S49] Shannon, C. E., "Communication in the presence of noise", Proceedings of the I. R. E., Vol. 37, 10-21, (1949).
- [S98] Suyama, A., "DNA chips - Integrated Chemical Circuits for DNA Diagnosis and DNA computers", To appear, (1998).
- [WQF+98] Wang, L., Q. Liu, A. Frutos, S. Gillmor, A. Thiel, T. Strother, A. Condon, R. Corn, M. Lagally, L. Smith, "Surface-based DNA computing operations: DESTROY and READOUT", 4th DIMACS Workshop on DNA Based Computers, University of Pennsylvania, June, 1998. DNA Based Computers, IV, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, (ed. H. Rubin), American Mathematical Society, (1999).
- [WLW+98] Winfree, E., F. Liu, Lisa A. Wenzler, N. C. Seeman, "Design and Self-Assembly of Two Dimensional DNA Crystals", Nature 394: 539-544, 1998. (1998).
- [WYS96] Winfree, E., X. Yang, N.C. Seeman, "Universal Computation via Self-assembly of DNA: Some Theory and Experiments", 2nd Annual DIMACS Meeting on DNA Based Computers, Princeton, June, 1996.
- [W98] Wood, D. H., "Applying error correcting codes to DNA computing", 4th DIMACS Workshop on DNA Based Computers, University of Pennsylvania, June, 1998. DNA Based Computers, IV, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, (ed. H. Rubin), American Mathematical Society, (1999).

Challenges and Applications for Self-Assembled DNA Nanostructures^{*}

John H. Reif^{**}, Thomas H. LaBean, and Nadrian C. Seeman

Duke University, New York University

Abstract. *DNA self-assembly* is a methodology for the construction of molecular scale structures. In this method, artificially synthesized single stranded DNA self-assemble into DNA crossover molecules (tiles). These DNA tiles have sticky ends that preferentially match the sticky ends of certain other DNA tiles, facilitating the further assembly into tiling lattices. We discuss key theoretical and practical challenges of DNA self-assembly, as well as numerous potential applications.

The self-assembly of large 2D lattices consisting of up to thousands of tiles have been recently demonstrated, and 3D DNA lattices may soon be feasible to construct. We describe various novel DNA tiles with properties that facilitate self-assembly and their visualization by imaging devices such as atomic force microscope. We discuss bounds on the speed and error rates of the various types of self-assembly reactions, as well as methods that may minimize errors in self-assembly. We briefly discuss the ongoing development of attachment chemistry from DNA lattices to various types of molecules, and consider application of DNA lattices (assuming the development of such appropriate attachment chemistry from DNA lattices to these objects) as a substrate for:

- (a) layout of molecular electronic circuit components,
- (b) surface chemistry, for example ultra compact annealing arrays,
- (c) molecular robotics; for manipulation of molecules using molecular motor devices.

DNA self-assembly can, using only a small number of component tiles, provide arbitrarily complex assemblies. It can be used to execute computation, using tiles that specify individual steps of the computation. In this emerging new methodology for computation:

-input is provided by sets of single stranded DNA that serve as nucleation sites for assemblies, and

^{*} A postscript version of this paper is at URL
<http://www.cs.duke.edu/~reif/paper/SELFASSEMBLE/selfassemble.ps>.

^{**} Contact address: Department of Computer Science, Duke University, Box 90129, Durham, NC 27708-0129. E-mail: reif@cs.duke.edu. John Reif is supported by Grants NSF/DARPA CCR-9725021, CCR-96-33567, NSF IRI- 9619647 and EIA-0086015, ARO contract DAAH-04-96-1-0448, and ONR contract N00014-99-1-0406. Nadrian C. Seeman is supported by Grants GM-29554 from the National Insitute of General Medical Sciences, ONR contract N00014-98-1-0093, DARPA/NSF grant NSF-CCR-97-2502, AFSOR contract F30602-98-C-0148 NSF grants CTS-9986512 and EIA-0086015.

-output can be made by the ligation of reporter strands of DNA that run through the resulting assembly, and then released by denaturing. DNA self-assembly can be used to execute massively parallel computations at the molecular scale, with concurrent assemblies that may execute computations independently. Due to the very compact form of DNA molecules, the degree of parallelism (due to distinct tiling assemblies) may be up to 10^{15} to possibly 10^{18} . We describe various DNA tiling assemblies that execute various basic computational tasks, such as sequences of arithmetic and logical computations executed in massively parallel fashion. We also consider extensions of these computational methods to 3D DNA tiling lattices and to assemblies that hold state.

1 Introduction to Tiling Self-Assemblies

1.1 Self-Assembly

This is a process involving the spontaneous self-ordering of substructures into superstructures.

Biological Self-Assembly. We take inspiration from the cell, which performs a multiplicity of self-assembly tasks, including the self-assembly of cell walls (via lipids), of microtubules, etc. Many of these biological self-assembly processes utilize the specificity of ligand affinities to direct the self-assembly. We will focus instead on self-assemblies whose components are artificially constructed tiles.

1.2 Domino Tiling Problems

These were defined by Wang [Wang61] (Also see the text [Grunbaum, et al, 87]). The input is a finite set of unit size square tiles, each of whose sides are labeled with symbols over a finite alphabet (the pads). Additional restrictions may include the initial placement of a subset of these tiles, and the dimensions of the region where tiles must be placed. Assuming arbitrarily large supply of each tile, the problem is to place the tiles, without rotation (a criterion that cannot apply to physical tiles), to completely fill the given region so that each pair of abutting tiles have identical symbols on their contacting sides. (See Figure 1.)

Domino tiling problems over an infinite domain with only a constant number of tiles was first proved by [Berger66] to be undecidable; proofs rely on constructions wherein tiling patterns simulate single-tape Turing Machines (see also [Berger66, Robinson71, Wang75]). Other results include reductions of NP-complete problems to finite-size tiling problems [LewisPapa81, Moore00]

1.3 Self-Assembly of Tiling Lattices

Domino tiling problems do not presume or require a specific process for tiling. However, we will presume the use of the self-assembly processes for construction of tiling lattices. In this self-assembly process, the preferential matching of tile

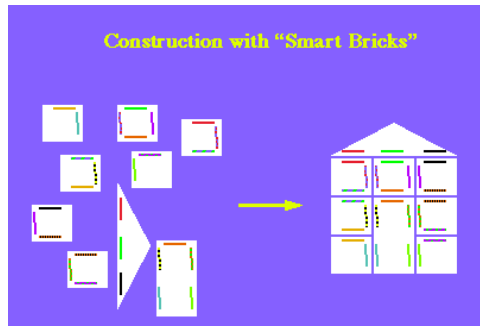


Fig. 1. A tiling assembly using ‘Smart Bricks’ with affinity between colored pads.

sides facilitates the further assembly into tiling lattices. The sides of the tiles are assumed to have some methodology for selective affinity, which we call *pads*. Pads function as programmable binding domains, which hold together the tiles.

Since domino tiling problems are undecidable (see Section 3), tiling self-assemblies can theoretically provide arbitrarily complex assemblies even with a constant number of distinct tile types. As a very simple example, it is possible to construct tiling assemblies with self-delimiting boundaries (e.g., rectangular boundaries of a given width w and length h), by use of a set of wh distinct tiles, with w distinct pad types on the bottom and top of a set of square tiles and a set of h pad types on the other sides of these tiles, and this can also be done with a constant number of distinct tiles. (See [Rothemund and Winfree, 2000b] for improved program-size complexity of self-assembled squares.)

Pad binding mechanisms for the preferential matching of tile sides can be provided by various methods:

- (i) *molecular affinity*, using for example hydrogen bonding of complementary DNA or RNA bases,
- (ii) *magnetic attraction*, e.g., pads with magnetic orientations constructed by curing the polymer/ferrite composites in the presence of strong magnetic fields, and also pads with patterned strips of magnetic orientations,
- (iii) *capillary force*, using hydrophobic/hydrophilic (capillary) effects at surface boundaries that generate lateral forces,
- (iv) *shape complementarity* (or conformational affinity), using the shape of the tile sides to hold them together.

There are a variety of distinct materials for tiles, at a variety of scales:

(a) Meso-Scale Tiling Assemblies have tiles of size a few millimeters up to a few centimeters. Whitesides at Harvard University has developed and tested multiple technologies [Zhao, et al, 98] [Xia et al, 98a,98b], [Bowden, et al 98], [Harder, et al 00] for meso-scale self-assembly, using capillary forces, shape complementarity, and magnetic forces (see <http://www-chem.harvard.edu/GeorgeWhitesides.html>). [Rothemund, 2000] also gave some meso-scale tiling assemblies using polymer tiles on fluid boundaries with pads that use hydrophobic/hydrophilic forces. A materials science group at the

U. of Wisconsin also tested meso-scale self-assembly using magnetic tiles (<http://mrsec.wisc.edu/edetc/selfassembly>). These meso-scale tiling assemblies were demonstrated by a number of methods, including placement of tiles on a liquid surface interface (e.g., at the interface between two liquids of distinct density or on the surface of an air/liquid interface).

(b) Molecular-Scale Tiling Assemblies have tiles of size up to a few hundred Angstroms. Specifically, DNA tiles will be the focus of our discussions in the following sections.

1.4 Goals and Organization of This Paper

The goal of this paper is describe techniques for self-assembly of DNA tiling arrays and applications of this technology, including DNA computation. We give in Section 2 a description of self-assembly techniques for DNA tilings and discuss applications in Section 3. Section 4 describes DNA tiling computations and their applications. Section 5 discusses the kinetics of self-assemblies and error control and Section 6 concludes the paper.

2 DNA Self-Assembly of DNA Tilings

2.1 DNA as a Construction Material

Nano-fabrication of structures in DNA was pioneered by the Seeman laboratory ([Seeman82, 94b, 96a]), who built a multitude of DNA nano-structures using DNA branched junctions [Seeman89, Wang91a, Du92]. These previous systems were flexible, so control over synthesis and proof of synthesis were both limited to the topological level, rather than the geometrical level (in contrast to the tiles described below). These DNA nano-structures included: DNA knots [Seeman93], Borromean rings [Mao97], a cube [Chen91], and a truncated octahedron [Zhang94] (reviewed in e.g. [Seeman98] [Seeman94b] and [Seeman96a]).

2.2 DNA Tiles Constructed from DX and TX Complexes

The building blocks in the tiling constructions to be discussed are branched DNA complexes, which we call *DNA tiles*, consisting of several individual DNA oligonucleotides that associate with well-defined secondary and tertiary structure (see [Winfree, et al 98] and below description of DX and TX tiles). These associate with well-defined secondary and tertiary geometric structure (which is much more predictable and less flexible than DNA nano-structures using DNA branched junctions). These complexes come in a number of varieties that differ from one another in the geometry of strand exchange and the topology of the strand paths through the tile. The branched DNA complexes used for tiling assemblies include the double-crossover (DX) and triple-crossover (TX) complexes. The DX and TX complexes consists of two (three, respectively) double-helices fused by exchange (crossover) of oligonucleotide strands at a number of separate

crossover points. Anti-parallel crossovers cause a reversal in direction of strand propagation through the tile following exchange of strand to a new helix. For example, DAO and DAE are double-crossover DX tiles with two anti-parallel crossovers¹. (See Figure 2.) .

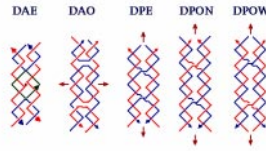


Fig. 2. Double crossover isomers.

DX complexes have been used successfully as substrates for enzymatic reactions including cleavage and ligation [Liu, Sha and Seeman,99]. The TX (see [LaBean,99]) TAO and TAE tiles, are similar except that they have three double-helices interlocked by exchange of oligonucleotide strands at four separate crossover points, two between the first pair of helices, two between the second (See the TAO in Figure 3 and the TAE in Figure 4. Both DX and TX motifs are useful for tiling assemblies; the DX (TX) complexes provide up to four (six, respectively) ssDNA pads [Liu, et al 99a] for encoding associations with neighboring tiles.

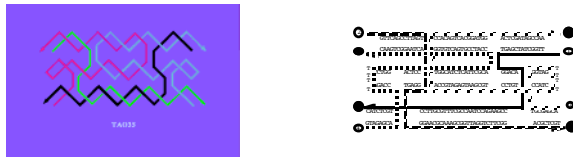


Fig. 3. The TAO tile and a Strand and Sequence Trace through the TAO Tile. DNA tiles are designed to contain several short sections of unpaired, single-strand DNA (ssDNA) extending from the ends of selected helices (often called 'sticky ends') that function as programmable binding domains, which are the *tile pads*.

¹ The structure of the TAE resembles the TAO in that it is constructed of three double-helices linked by strand exchange, however, it contains an Even (rather than Odd) number of helical half-turns between crossover points. Even spacing of crossovers allows reporter strands (shown in black) to stretch straight through each helix from one side of the tile to the other. These three horizontal reporter segments are used for building up a long strand which records inputs and outputs for the entire assembly computations. A 3D confirmation of the TAE tile has been rendered by Brendon Murphy at Duke University; see <http://www.duke.edu/~bkm2/taehtml/present.html>.



Fig. 4. The TAE Tile.

2.3 DNA Tiling Lattices

Tile assemblies, or simply tilings, can be defined as superstructures or lattices built up from smaller, possibly repetitive, component structures. Individual tiles interact by annealing with other specific tiles via their ssDNA pads to self-assemble into desired superstructures. These lattices can be either:

- (a) *non-computational*, containing a fairly small number of distinct tile types in a repetitive, periodic pattern; or
- (b) *computational*, containing a larger number of tile types with more complicated association rules which perform a computation during lattice assembly.

These DNA self-assembly procedures generally will be described as occurring in two distinct stages:

- (i) annealing of ssDNA into tiles; and
- (ii) assembly of tiles into superstructures.

However, direct assembly of DNA lattices from component ssDNA is also possible, and has in fact already been demonstrated for non-computational DNA lattices described below.

2.4 Two Dimensional DNA Tiling Assemblies

Recently Winfree and Seeman have demonstrated the use DX tiles to construct 2D periodic lattices consisting of upto a hundred thousand DX units [Winfree, et al 98] as observed by atomic force microscopy². The surface features are readily programmed [Winfree et al., 98; Liu et al., 99; Mao, et al 99]. (See Figure 5.) In addition, [LaBean et al, 99] constructed produced tiling arrays (see Figure 6) composed of DNA triple crossover molecules (TX); these appear to assemble at least as readily as DX tiles. These tiling assemblies had no fixed limit on their size. [Reif97] introduced the concept of a *nano-frame*, which is a self-assembled nanostructure that constrains the subsequent timing assembly (e.g., to a fixed size rectangle). Alternatively, a tiling assembly might be designed to be *self-delimitating* (growing to only a fixed size) by the choice of tile pads that essentially 'count' to their intended boundaries in the dimensions to be delimited.

² An atomic force microscope [AFM] is a mechanical scanning device that provides images of molecular structures laying on a flat 2D plate.

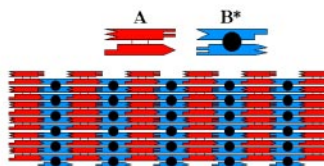


Fig. 5. AB* Array. Lattice formed from two varieties of DX, one containing an extra loop of DNA projecting out of the lattice plane, facilitating atomic force microscope imaging of the lattice.

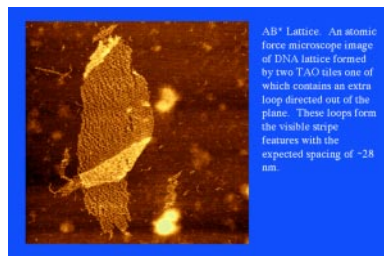


Fig. 6. A non-computational DNA tiling formed by sets of two types of TAO tiles.

2.5 Three Dimensional DNA Tiling Assemblies

There are a number of possible methods for constructing 3D periodic (non-computational) tilings. For example, stable tiling arrays with well-defined helices that come out of the plane (e.g., the TX tiling array constructed in [LaBean et al, 99]) may lead to ways to provide 3D tiling assemblies.

3 Applications of Non-computational DNA Tiling Arrays

We now identify some further technological impacts of non-computational DNA assemblies; particularly their use as substrates for surface chemistry and molecular electronics, robotics. Many of these applications are dependant on the development of the appropriate attachment chemistry between DNA and the molecules attached to the arrays.

3.1 Application to Layout of Molecular-Scale Circuit Components

Molecular-scale circuits have the potential of replacing the traditional micro-electronics with densities up to millions of times current circuit densities. There have been a number of recent efforts to design molecular circuit components ([Petty et al 95] [Aviram,Ratner98]). Tour at Rice Univ. in collaboration with Reed at Yale have designed and demonstrated [Chen et al 99] organic molecules

(see Figure 7) that act as conducting wires [Reed et al.97],[Zhou99] and also rectifying diodes (showing negative differential resistance (NDR), and as well as [CRR+,99], [RCR+,00], and have the potential to provide dynamic random access memory (DRAM) cells. These generally use $\sim 1,000$ molecules per device, but they have also addressed single molecules and recorded current through single molecules [BAC+96], [RZM+97]. These molecular electronic components make conformational changes when they do electrical switching. One key open problem in molecular electronics is to develop molecular electronic components that exhibit restoration of a signal to binary values; one possible approach may be to make use of multi-component assemblies that exhibit cooperative thresholding.

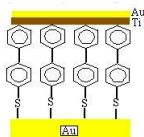


Fig. 7. The Tour-Reed molecular electronic diode.

The Molecular Circuit Assembly Problem: This key problem is to develop methods for assembling these molecular electronic components into a molecular scale circuit. Progress in the molecular circuit assembly problem could have revolutionary impact on the electronic industry, since it is one of the key problems delaying the development of molecular-scale circuits.

Top-down techniques versus bottom-up approaches. The usual approach of laying out circuits by top-down techniques (e.g., lithography) may not be practical at the molecular scale; instead bottom-up approaches (e.g., self-assembly) may need to be used. Hence this may be a key area of application of DNA tiling assemblies. There are a number of possible methods for the selective attachment of the molecular electronic components to particular tiles of the DNA tiling array, using annealing.

(i) *linking chemistry between DNA and molecular electronics.* Tour and Bunz recently prepared DNA-linked systems where the DNA could serve as a selective assembly glue for device configurations [WST+00].

(ii) *The use of gold beads.* In this approach, DNA strands attached to the gold beads can hybridize at selected locations of the arrays, and the molecular electronics components may self-assemble between the gold beads.

Also, DNA lattices may be useful as a foundation upon which to grow nano-scale gold wires. This might be done by depositions of gold from colloid onto nano-spheres immobilized on DNA tiling lattices. (See Figures 8 and 9.)

Molecular probe devices may be used to test the electrical properties of the resulting molecular circuit attached to the DNA tiling array. *Computational* lattices (as opposed to regular, non-computational lattices), may also be employed

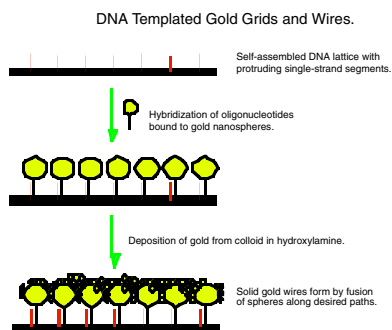


Fig. 8. Diffusion of gold on beads to form molecular-scale gold wires.

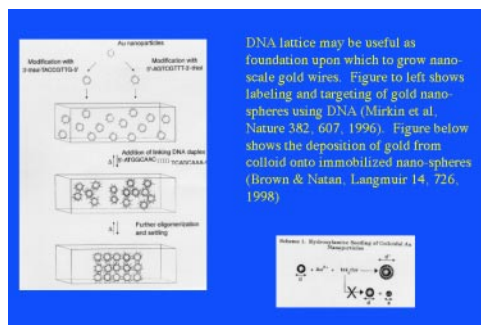


Fig. 9. A scheme for molecular-scale gold wires.

to provide for the layout of highly complex circuits, e.g., the layout of the electronic components of an arithmetic unit.

(Other related approaches for positioning of molecular electronic molecules without lithography include that of [Bumm, et al., 1999, 2000], which describes the use of directed self-assembly of molecular terrace structures in organic monolayers.)

3.2 Application to Surface Chemistry and Impact on Biotechnology

One intriguing application for DNA lattices is their use as an attachment substrate for an array of DNA strands, using hybridization with single stranded DNA on individual tiles. This has a number of applications that impact DNA computations (e.g., see Brockman, et al 98)[Smith,98]), as well as more general biotechnology:

- (a) It may provide a dramatic miniaturization of the DNA chip technology (a technology that we have noted above might be used for I/O in DNA computations, among other applications), to molecular scale aspect widths.
- (b) It may provide a dramatic miniaturization of DNA computation methods using surface chemistry [Corn, et al 99], again to molecular scale aspect widths.

Two dimensional DNA tiling lattices may in the future be self-assembled with each of the tiles modified to have dangling *hybridization* strands, which will be single stranded, as used by [Liu et al., 1999]. These hybridization strands will be thus assembled in a very regular, dense fashion, and may have sequences determined by the computational tiling. The differential hybridization of these hybridization strands may be demonstrated with fluorescence tagged complementary DNA.

3.3 Tiling Assemblies with Molecular Motors

Several types of molecules are known to couple chemical energy to the generation mechanical force, thereby functioning as molecular motors. Possible schemes for molecular motors include:

(a) Re-Engineering Biological Molecular Motors. Cells make use of a variety of such motor-like devices in processes such as mitosis. The best characterized of these fall into three categories.

(i) *ATP synthase and ADP* acts as a rotary motor, coupling proton flux through a membrane with the phosphorylation of ADP to ATP. The F1 component of ATP synthase can also be run in reverse, coupling hydrolysis of one ATP molecule to 120 B0 of rotation about the motor axis.

(ii) *Myosin* acts as a molecular running machine, skipping many steps along an actin filament with each molecule of ATP consumed. All of these motors are modular and can be re-engineered to accomplish linear or rotational motion of essentially any type of molecular component.

(iii) *Kinesin* acts as a molecular walking machine, translocating itself (and any attached components) in step-wise fashion along a microtubule. Each step along the microtubule consumes one ATP molecule.

Construction of these biological molecular motors and their linking chemistry to DNA arrays. These motors are composed of proteins with well known transcription sequences. There are also well known proteins (binding proteins) that provide linking chemistry to DNA. Hence it seems feasible that these molecular motors and attached linking elements may be synthesized from sequences obtained by concatenation of these transcription sequences. [Bachand, et al., 1999 and 2000], describes a biomolecular motor constructed of expressed ADP protein [Montemagno, et al, 1998 and 1999] with an attached [Soong, et al, 1999] silicon arm.

(b) DNA Motors. The Seeman laboratory made a DNA construction of a mechanical device capable of controlled movement [Mao, et al 99a, Seeman, et al 99]. This device consists of two DX molecules connected by a DNA double helix that contains a segment of DNA that can be converted to the left-handed Z-DNA structure. In B-promoting conditions, the two unconnected helices of the device are on the same side of the connecting helix, but they are on opposite sides in Z-promoting conditions. This results in an apparent rotary motion of about a half-revolution, leading to atomic displacements ranging from 2 to 6 nm, depending on the location of the atom relative to the axis of the stationary helix. This motion has been demonstrated by fluorescent resonance energy transfer

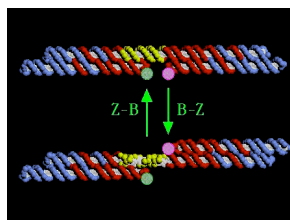


Fig. 10. A prototype DNA nanomechanical device.

(FRET). It is important to point out that the device based on the B-Z transition is only a prototype that was used to learn how to characterize motion in DNA systems. It lacks programmability, except to the limited extent that one can orient the two DX molecules at a variety of relative torsion angles in the B-state. Thus, all of the molecules must be in either the B-state or in the Z-state, assuming one has robust chemical control.

These molecular motors might be combined with the 2-D arrays, to achieve an array of devices (This has not been possible to do with the DX system, since it is most convenient there for the pivoting part of the system to point normal to the array. However, the TX system does not have this difficulty.). As an example, an array with attached kinesin may provide for the movement of objects across the surface of a two dimensional tiling array, similar to a conveyer belt, and this may be the basis of a transport system (*a molecular conveyer belt*) for molecular objects.

Programmable Sequence-Specific Control of DNA Mechanical Motion. However, such an array of molecular motors would be more useful if they can be selectively controlled. Such a system would lead to the ability to manipulate specific molecules and more generally, to do chemistry at chemically identical but spatially distinct sites. Because it couples a series of distinct structural states with programmability, such a system offers the potential of direct route to molecular robotics. The Seeman laboratory is developing a related system based on the paranemic crossover (PX) system, which leads to sequence-specific nanomechanical motion. It may be switched readily between two discrete states, PX or JX, in which the helices at one end of the molecule reverse positions in the transition between states. An array of these molecular devices would contain individually programmed PX/JX molecules, whose conformational state would be amenable to specific reversal (or not, depending on the program) from cycle to cycle. A DNA array with programmability of this sort may offer a mechanism to do DNA computation of arrays whose elements (the tiles) hold state, as discussed in the next section.

4 Computation by Self-Assembled Tilings

4.1 DNA Computation

In his seminal paper on molecular computation [Adleman94], Adleman demonstrated the use of recombinant DNA techniques for solving a small combinatorial search problem. This work spawned considerable further work in DNA computation (See survey of [Reif, 1998]). However, one difficulty with such methods for DNA computation is the number of laboratory procedures, each time consuming and error-prone, grows with the size of the problem.

4.2 Computation by DNA Self-Assembly

We now focus on another approach: computation by self-assembly. In this case Self-assembly is the spontaneous self-ordering of substructures into superstructures driven by annealing of Watson-Crick base-pairing DNA sequences. Computation by self-assembly entails the building up of superstructures from starting units such that the assembly process itself performs the actual computation. Adleman made use of a simple form of computation by self-assembly in his original experiment [Adleman94]: instead of blindly generating all possible sequences of vertices; instead, the oligonucleotide sequences and the logic of Watson-Crick complementarity guide the self-assembly processes so that only valid paths are generated. [Winfree95] generalized this approach to two-dimensional (2D) self-assembly processes and showed that computation by self-assembly is Turing-universal (also see prior Turing-universal results for tiling [Buchi62, Wang63, 75, Berger66, Robinson71] discussed below).

[Winfree96, Eng97] proposed self-assembly of linear, hairpin, and branched DNA molecules to generate regular, bilinear, and context-free languages, respectively. [Winfree96], Jonoska et al [Jonoska97, Jonoska98], and [Lagoudakis and LaBean,99] all proposed the use of self-assembled DNA nanostructures to solve NP complete combinatorial search problems (but the scale is limited to only moderate size problems at best).

Programming Self-Assembly of DNA Tilings. Programming DNA self-assembly of tilings amounts to the design of the pads of the DNA tiles (recall these are sticky ends of ssDNA that function as programmable binding domains, and that individual tiles interact by annealing with other specific tiles via their ssDNA pads to self-assemble into desired superstructures). The use of pads with complementary base sequences allows the neighbor relations of tiles in the final assembly to be intimately controlled; thus the only large-scale superstructures formed during assembly are those that encode valid mappings of input to output. Consequently, the difficulty mentioned previously (with respect to DNA computation) has been addressed: rather than implementing a DNA computing algorithm using a sequence of multiple laboratory procedures, our approach essentially uses only four:

- (i) mixing the input oligonucleotides to form the DNA tiles,
- (ii) allowing the tiles to self-assemble into superstructures,

- (iii) ligating strands that have been co-localized, and
- (iv) then performing a single separation to identify the correct output.

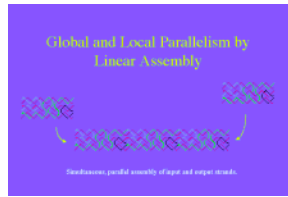


Fig. 11. Global and Local Assembly Parallelism. *Local tile association rules dictate that only valid computational lattices are able to form regardless of temporal ordering of binding events.*

4.3 Massively Parallel Computation by Tiling

The massive parallelism inherent in DNA-based computers has, since its inception, driven thinking in the field. In computation by self-assembly, parallelism reveals itself in many ways. Each superstructure may contain information representing a different calculation (*global parallelism*). Growth on each individual superstructure may occur at many locations simultaneously *local parallelism*. (See Figure 4.2)

The *depth* of a tiling superstructure is the maximum number of self-assembly reactions experienced by any substructure (the depth of the graph of reaction events), and the *size* of a superstructure is the number of tiles it contains. Likewise for the number of layers. A superstructure consisting of an array of $n \times m$ tiles, where $n > m$, is said to have m layers. Again, although it needs further study, tiling systems with low depth, small size, and few layers are considered desirable, motivating the search for efficient computations performed by such systems. Tiling systems that produce only superstructures with k layers, for some constant k , are said to use *linear self-assembly*. As an example, the two tiling systems for addition discussed in [LaBean, et al 99] for n -bit sums produce linear superstructures that are arrays of size $3 \times n$ and $1 \times n$, but known tiling systems for multiplication produce $n \times n$ for n -bit products [Winfree99a], and hence are not linear.

[Reif97] described DNA self-assembly methods of linear size and small depth to solve a number of fundamental problems (e.g., prefix computation, permutation, certain integer arithmetic operations, finite state automata simulation, and string fingerprinting) that form the basis for the design of many parallel algorithms. Furthermore, these elementary operations can be combined to perform more complex computations, such as bitonic sorting and general circuit evaluation in $O(\log n)$ experimental steps.

4.4 The Speed of Computing via DNA Tiling Assemblies (Compared with Silicon-Based Computing)

The speed of DNA tiling assemblies is limited by the annealing time, which can be many minutes, and can be 10^{11} slower than a conventional computer. A DNA computation via self-assembly must take into account the fact that the time to execute an assembly can range from a few minutes up to hours. Therefore, a reasonable assessment of the power of DNA computation must take into account both the speed of operation as well as the degree of massive parallelism. Nevertheless, the massive parallelism (both within assemblies and also via the parallel construction of distinct assemblies) possibly ranging from 10^{15} to 10^{18} provides a potential that may be advantageous for classes of computational problems that can be parallelized.

4.5 String-Tiles: A Mechanism for Small-Depth Tiling

An approach for small-depth computations is to compress several tile layers into single tiles, so that the simplest form of linear self-assembly suffices. Linear self-assembly schemes for integer addition were first described by [Reif97]; in this scheme each tile performed essentially the operation of a single carry-bit logic step. This linear self-assembly approach works particularly well when the topology and routing of the strands in the DNA tiles is carefully considered, leading to the notion of string tiles. The concept of string tile assemblies derives from Eng's observation that allowing neighboring tiles in an assembly to associate by two sticky ends on each side, he could increase the computational complexity of languages generated by linear self-assembly. [Winfree99a] showed that by allowing contiguous strings of DNA to trace through individual tiles and the entire assembly multiple times, surprisingly sophisticated calculations can be performed with 1-layer linear assemblies of string tiles. The TAE tiles recently developed by [LaBean, et al 99] are particularly useful as string tiles.

4.6 Input/Output to Tiling Assemblies Using Scaffold and Reporter Strands

Input and output are critical to the practical use of DNA-based computing.³ Winfree [Winfree95] used the first and last layers of the assembly for input

³ There are a number of known methods for synthesis of DNA from sequence information provided by conventional media, (e.g., by PCR, restriction cutting, or sequencing). but these provide limited I/O rates. While it is not the purpose of this work to develop improved biotechnology techniques for DNA sequence synthesis and sequencing, we note this is a key goal for an entire sector of the biotechnology industry, and therefore it is likely that the capabilities will significantly improve in the future. One of the most promising is the possible use of large arrays of (perhaps up to a 1,000) DNA chips. Each DNA chip can have up to 100,000 individually addressed locations, each capable of light-directed chemical synthesis of DNA strands. Similar technology, using fluorescently tagged DNA hybridized to DNA chip anneal-

and output, respectively. The TAO and TAE tiles have an interesting property, namely that certain distinguished single stranded DNA (to be called scaffold and reporter strands, respectively) wind through all the tiles of a tiling assembly. This property provides a more sophisticated method for input and output of DNA computations in string tiling assemblies:

(a) Input via Scaffold Strands: We take as input the scaffold strands and which encode the data input to the assembly computation. (See Figure 3.) They are long DNA strands capable of serving as nucleation points for assembly. Pre-formed, multimeric scaffold strands are added to the hybridization/annealing mixture in place of the monomeric oligo corresponding to the tile's reporter segment. The remaining portion of the component ssDNA comprising the tiles are also added. In the resulting annealing process, tiles assemble around the scaffold strand, automatically forming a chain of connected tiles which can subsequently be used as the input layer in a computational assembly.

(b) Output via Reporter Strands: After ligation of the tiling assembly (this joins together each tiles segments of the reporter strands), the reporter strand provides an encoding of the output of the tiling assembly computation (and typically also the inputs). Note this input/output can occur in parallel for multiple distinct tiling assemblies. Finally, the tiling assembly is disassembled by denaturing (e.g., via heating) and the resulting ssDNA Reporter Strands provide the result (these may be used as scaffold strands for later cycles of assembly computation, or the readout may be by PCR, restriction cutting, sequencing, or DNA expression chips).

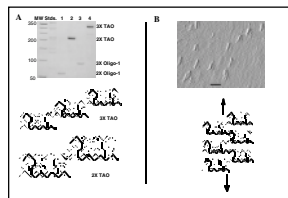


Fig. 12. Use of scaffold strands in assemblies.

4.7 One Dimensional DNA Tiling Computations or Parallel Arithmetic

We now outline (See Figure 13.) procedures for using the string tiles described above that self-assemble into linear tiling assemblies to perform massively parallel arithmetic. [LaBean, et al 99] describes string tile systems that compute

ing arrays, can be used for output of DNA information to conventional media. This could potentially provide up to approximately a factor of 100,000,000 parallelism in I/O. See [Reif, et al, 2000] for a discussion of these techniques and for methods for error-correction methods for DNA chip synthesis).

binary number addition (where the binary numbers are encoded by strands of DNA) by using two distinct sets of sticky-ends between adjacent tiles in the assembly to effectively communicate the values of the carry-bits. (They can also be used for computation of bit-wise XOR of Boolean vectors encoded by strands of DNA.) The assemblies result in the appending of these strands to the addition sums. For computations on specific inputs, these procedures make use of the scaffold strands mentioned above. The inputs will be self-assembled strands of DNA composed of sequences DNA words encoding the pairs of binary numbers to be summed. Otherwise, the input tiles can be (using known techniques uses for the assembly of combinatorial libraries of DNA strands) randomly assembled and thereby generate a molecular look-up table in which each reporter strand encodes the random inputs and resultant outputs of a single calculation. After denaturing the assemblies back to individual strands, one may sample the resulting reporter strands to verify the outputs are correctly computed. A sufficient number of DNA tile molecules will provide full coverage of all possible n-bit input strings. Such look-up tables may be useful as input for further computations as they represent a unique library of sequences with a complex structural theme. An experimental demonstration of this tiling is described in [Mao, LaBean, Reif, and Seeman, 2000].

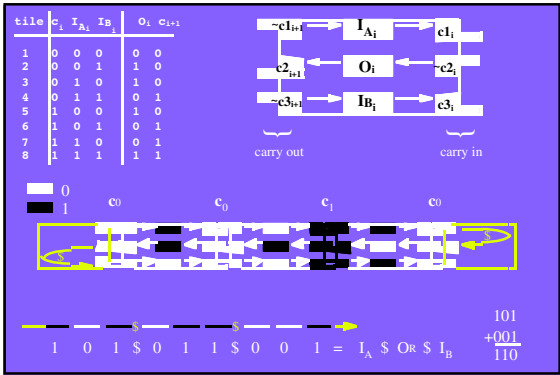


Fig. 13. String Tile Addition with TAE Building Blocks. *Upper left shows the truth table for addition; one tile type will be required for each row in the table. Upper right shows a schematic of a tile including the sequence positions for encoding values for input bits (I_{Ai} and I_{Bi}), the output bit (O_i), and the carry bit values on the tile's sticky-ends. The center schematic shows a five tile complex carrying out the addition of two 3-bit numbers. Arrows indicate the trace of the reporter strand as it winds through the entire assembly three times. The left and right extreme tiles act to reroute the reporter strand back through the lattice. The two corner tiles have been successfully built and shown to properly associate with one another.*

4.8 Two Dimensional DNA Tiling Computations

In the immediate future, it may be possible to extend the one dimensional DNA tiling assembly methods to two dimensional tilings, and to demonstrate these methods experimentally. One interesting goal is integer multiplication. The most direct and relatively straightforward way is to multiply via repeated additions and bit shifts, applying known VLSI systolic array architecture designs for integer multiplication. This would require a two dimensional $n \times n$ tiling assembly, with some increased complexity over the linear assembly for integer addition. Two dimensional computational tilings may also be used to do logical processing. [Lagoudakis and LaBean,99] proposed a 2D DNA self-assembly for Boolean variable satisfiability, which uses parallel construction of multiple self-assembling 2D DNA lattices to solve the problem. Such methods for solving combinatorial search problems do not scale well with the input size (the number of parallel tiling assemblies grows exponentially with the number of Boolean variables of the formula). However, similar constructions may be used for evaluating Boolean queries and circuits in massively parallel fashion, for multiple input settings of the input Boolean variable, and in this context it may be appropriate to consider the Boolean formulas to be of fixed size.

4.9 Three Dimensional DNA Tiling Computations

There are number of possible methods for executing computations experimentally on 3D DNA lattices, providing computations with (implicit) data movement in three dimensions. Matrix inner product might be executed by a three dimensional computational tiling by applying known VLSI systolic array architecture designs for matrix inner product. Another possible three dimensional computational tiling is that of the time-evolution (time is the third dimension of the tiling) of a two dimensional cellular automata (e.g., a two dimensional cellular automata simulation of fluid flow).

4.10 Recycling of the Component ssDNA

In principle, after a cycle of tiling assembly and disassembly, the component ssDNA comprising these tiles can simply be separated (for example, by magnetic bead separation), and reused for further cycles of assembly computations. For non-computational tiling assemblies, heat cycling has exactly this effect, and a few such cycles does not appear to result in noticeable degradation of the component ssDNA. Such re-cycling methods might be used for computational tiling assemblies, though there seems no apparent reason why there would be more degradation in this case of computational assemblies. (If ssDNA degradation does occur over many cycles, one may need to consider methods for repair, such as enzymatic repair of component ssDNA.) Another possibly serious issue is that of contaminating the 'result' ssDNA holding the results of successive computations; for example these result ssDNA strands may be contaminated by partial products due to incomplete ligation of the tiling assembly. In principle,

these errors might be corrected by additional steps that provide purification of the result ssDNA (e.g., separate out the ssDNA of the correct mass or having specific primers at the start and end of the calculation to amplify the correct answer by PCR) using conventional recombinant DNA techniques. Varied DNA backbones [e.g., Neilsen, 1995] may prove another approach to increasing the covalent stability

4.11 Arrays of Finite State Machines

A DNA array of motors, as described in the previous section, may offer a mechanism to do DNA computation of arrays whose elements (the tiles) hold state. That is, the DNA assemblies may be able to simulate a parallel computing model known as cellular automata, which consist of arrays of finite state automata, each of which holds state. The transitions of these automata and communication of values to their neighbors might be done by conformal (geometry) changes, again using this programmability. There are numerous examples of 1 D (2 D, respectively) cellular automata that can do computations for which tiling assemblies would have required a further dimension (for example, integer multiplication in one dimension instead of two).

5 The Kinetics and Error Control in Self-Assembled Tiling Assemblies

5.1 Kinetics of Self-Assembled Tiling Assemblies

In spite of an extensive literature on the kinetics of the assembly of regular crystalline lattices, the fundamental thermodynamic and kinetic aspects of self-assembly of tiling assemblies are not yet well understood. For example, it is not yet known the affect of distinct tile concentrations and different relative numbers of tiles, and the possible application of Le Chatelier's principle.

Winfree [W98] developed computer simulation of tiling self-assemblies. This software makes a discrete time simulation of the tiling assembly processes, using approximate probabilities for the insertion or removal individual tiles from the assembly. These simulations provide an approximation to the kinetics of self-assembly chemistry. Using this software as a basis, Guangwei Yuan at Duke developed improved simulation software with a Java interface (<http://www.cs.duke.edu/~yuan/gw/project/test.HTML>) for a number of example tilings, such as string tilings for integer addition and XOR computations. This simulation software was recently speed up by use of an improved method for computing on/of likelihood, as suggested by Winfree.

The meso-scale tiling experiments described in Section 1 have used mechanical agitation with shakers to provide a temperature setting for the assembly kinetics (that is, a temperature setting is made by fixing the rate and intensity of shaker agitation). These meso-scale tilings also have potential to illustrate fundamental thermodynamic and kinetic aspects of self-assembly chemistry.

5.2 Error Control in Self-Assembled Tiling Assemblies

As stated above, two dimensional self-assembled *non-computational* tilings have been demonstrated (and imaged via atom force microscopy) that involve up to a hundred thousand tiles. Certain of these appear to suffer from relatively low defect rates, perhaps in the order of less than a fraction of a percentage or less. The factors influencing these defect rates are not yet well understood and there are no known estimates on the error-rates for self-assembled *computation* tilings, since such tilings have been achieved only very recently and have only been done on a very small scale (error rates appear to be less than 5% [Mao et al 00]). There is reason (see the construction of a potential assembly blockage described in [Reif, 98]) to believe that in computational tilings, defect errors may be more prevalent; and moreover, they can have catastrophic effects on the computation. Experiments need to be done to determine the error rates of the various types of self-assembly reactions, both computational and non-computational.

There are a number of possible methods to decrease errors in DNA tilings. It is as yet uncertain which methods will turn out to be effective and it is likely that a combination of at least a few of the following methods will prove most effective.

(a) Error Control by Annealing Temperature Variation. This is a well known technique used in hybridization and also crystallization experiments. It is likely that this will provide some decrease in defect rates at the expense in increased overall annealing time duration. In the context of DNA tiling lattices, the parameters for the temperature variation that minimize defects have not yet been determined.

(b) Error Control by Improved Sequence Specificity of DNA Annealing. The most obvious methodology here is to improve the choice of the DNA words used for tile pads (that is, to decrease the likelihood of incorrect hybridizations from non-matching pads). DNA word design software for DNA tiles was developed by Winfree. This software has recently been improved by Horatiu Voicu at Duke University (see <http://www.cs.duke.edu/~hvoicu/app.html>) to include more realistic models of DNA hybridization, and was also speeded up by use of a technique of Rajasakaran and Reif [RR95] known as nested annealing. Another possible approach is to use the observation and experimental evidence of [Herschlag 91] that stressed DNA molecules can have much higher hybridization fidelity (sequence specificity) than a relaxed molecule. This would entail redesigning DNA tiles, so their pads are strained single stranded loop segments with higher sequence specificity, or by the use of stressed DNA motifs known as “PX dumbbells” [Shen, Z. Ph.D. Thesis, NYU, 1999].

(c) Error Control by Redundancy. There are a number of ways to introduce redundancy into a computational tiling assembly. One simple method that can be developed for linear tiling assemblies, is to replace each tile with a stack of three tiles executing the same function, and then add additional tiles that essentially ‘vote’ on the pad associations associated with these redundant tiles. This results in a tiling of increased complexity but still linear size. This error resistant design

can easily be applied to the integer addition linear tiling described above, and similar redundancy methods may be applied to higher dimension tilings.

(d) Error Control by Free Versus Step-wise Assembly. Self-assembly may be restricted so that certain assembly reactions can proceed only after others have been completed (*serial (or step-wise) self-assembly*). Alternatively, self-assembly reactions may be limited by no such restrictions (*free self-assembly*). It is expected, but unproven, that free self-assembly is faster than serial self-assembly. [Reif, 98] suggested the use of serial self-assembly to decrease errors in self-assembly. There is not yet any experimental data on the error rates of self-assembly reactions and error control/repair of ‘self-assembly’ versus ‘serial-assembly’. To decrease the human effort in serial assembly, assembly steps might be executed automatically with the use of a robotic machines (E.g., the Nanogen machine, which employs a chip that contains DNA sequences above electrodes. Tile components hybridized to these DNA sequences can be released in sequence by making the electrode sufficiently negative.).

(e) Use of DNA Lattices as a Reactive Substrate for Error Repair. DX complexes and lattices have been used successfully as substrates for enzymatic reactions including restriction cleavage and ligation of exposed hairpins attached to the tiles [Liu99a]. One approach is the use of DNA lattices to execute a broader class of reactions. For example, if restriction enzymes, topoisomerases or site-specific recombinases can be shown to operate on exposed portions of the DNA lattices, then it may be possible to modify the topology and geometry of the DNA lattice. This may aid in the DNA tiling computations described above, for example by providing mechanisms for error repair in DNA tiling computations. (Note: as mentioned above, this approach may also be of use for recycling of the component ssDNA for the next computation cycles.)

6 Conclusion

We have discussed the potential advantages of self-assembly techniques for DNA computation; particularly the decreased number of laboratory steps required. We also discussed the potential broader technological impacts of DNA tiling lattices and identify some further possible applications. The chief difficulties are that of error control and predictable kinetics, as described in the previous section. Nevertheless, the self-assembly of DNA tilings seems a very promising emerging method for molecular scale constructions and computation.

References

1. Adleman, L., Molecular Computation of Solution to Combinatorial Problems, Science, 266, 1021, (1994).
2. Aviram and M. Ratner (Eds), Molecular Electronics: Science and Technology, Annals of the New York Academy of Sciences, New York, Vol. 852 (1998).

3. Bachand, G.D., and Montemagno, C.D. Constructing organic/inorganic NEMS devices powered by bio-molecular motors. *Biomedical Microdevices*, in press, (2000).
4. Bachand, G. D., and Montemagno, C. D. Constructing biomolecular motor-powered, hybrid NEMS devices. *Technical Proceedings of the International Symposium on Microelectronics and Micro Electro Mechanical Systems (MICRO/MEMS)*. Queensland, Australia, (1999).
5. Berger, R. The Undecidability of the Domino Problem, *Memoirs of the American Mathematical Society*, 66 (1966).
6. S. Blawas, T. F. Oliver, M. C. Pirrung, and W. M. Reichert, Step-and-repeat Photopatterning of Protein Features Using Caged-biotin-BSA: Characterization and Resolution, *Langmuir*, 14, 4243 (1998).
7. Bowden, N.; Brittain, S.; Evans, A. G., Hutchinson, J. W. and Whitesides, G. M. Spontaneous formation of ordered structures in thin films of metals supported on an elastomeric polymer, *Nature* 1998,393, 146-149.
8. Buchi, J.R Turing Machines and the Entscheidungsproblem, *Mathematische Annalen*, 148, 201-213, (1962).
9. Brockman, J.M.; Frutos, A.G. and Corn, R.M. A Multi-Step Chemical Modification Procedure to Create DNA Arrays on Gold Surfaces for the Study of Protein-DNA Interactions with Surface Plasmon Resonance Imaging, *J. Am. Chem. Soc.*, 121 8044-8051 (1999).
10. Bumm, L. A. and P. S. Weiss, Positioning Atoms and Molecules without Lithography: Self-Assembly and Directed Assembly, invited manuscript in preparation for *Superlattices and Nanostructures*, (2000). <http://stm1.chem.psu.edu/~psw/papers/DirectAsmbyJACS99.pdf>
11. Bumm, L. A., J. J. Arnold, L. F. Charles, T. D. Dunbar, D. L. Allara, and P. S. Weiss, Directed Self-Assembly to Create Molecular Terraces with Molecularly Sharp Boundaries in Organic Monolayers, *Journal of the American Chemical Society* 121, 8017 (1999). (ABSTRACT)
12. J. Chen, M. A. Reed, A. M. Rawlett, and J. M. Tour, Observation of a Large On-Off Ratio and Negative Differential Resistance in an Molecular Electronic Device, *Science*, Vol 286, 19, Nov. 1999, p 1550-1552.
13. Clelland, C.T., Risca, V., and C. Bancroft. Genomic Steganography: Amplifiable Microdots. Under review by *Nature*, 1999.
14. E. Coven, N. Jonoska: DNA Hybridization, Shifts of Finite type and Tiling of the Integers, (to appear in: *Words, Sequences, Languages: where computer science and linguistics meet*, published by Kluwer, edited by Carlos Martin-Vide).
15. Du, S.M., S. Zhang and N.C. Seeman, DNA Junctions, Antijunctions and Mesojunctions, *Biochem.*, 31, 10955-10963, (1992).
16. Eng, T., Linear DNA self-assembly with hairpins generates the equivalent of linear context-free grammars, 3rd DIMACS Meeting on DNA Based Computers, Univ. of Penn., (June, 1997).
17. Faulhammer, D., Cukras, A.R., Lipton, R.J. and Landweber, L.F., Molecular computation: RNA solutions to chess problems, *Proc. Nat. Acad. Sci. (USA)* 97, 1385-1389 (2000).
18. Fu, T.-J., and N.C. Seeman, DNA Double Crossover Structures, *Biochemistry*, 32, 3211-3220, (1993).
19. Garey, M. R., and D. S. Johnson *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H Freeman and Company, page 257, (1979).

20. Gehani, A and J.H. Reif, Microflow Bio-Molecular Computation, 4th DIMACS Workshop on DNA Based Computers, University of Pennsylvania, June, 1998. DNA Based Computers, IV, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, (ed. H. Rubin), American Mathematical Society, 1999. Appeared also in Biosystems, Journal of Biological and Informational Processing Sciences, Vol. 52, Nos. 1-3, (ed. By L. Kari, H. Rubin, and D. H. Wood), pp 197-216, (1999). www.cs.duke.edu/~reif/paper/geha/microflow.ps
21. Gehani, A., T. H. LaBean, and J.H. Reif, DNA-based Cryptography, 5th DIMACS Workshop on DNA Based Computers, MIT, June, 1999. DNA Based Computers, V, DIMACS Series in Discrete Mathematics and Theoretical CS (ed. E. Winfree), American Mathematical Society, 2000. www.cs.duke.edu/~reif/paper/DNACrypt/crypt.ps
22. Gillmor, S.D., Liu, Q., Thiel, A.J., Condon, A.E., Corn, R.M., Smith, L.M. and Lagally, M.G. Hydrophobic/Hydrophilic Patterned Surfaces to Create DNA Arrays, to be submitted (to Langmuir), 1999.
23. Grunbaum, S., Branko, and G.C. Shepard, Tilings and Patterns, H Freeman and Company, Chapter 11, (1987).
24. Harada, K. and Orgel, L.E., Unexpected substrate specificity of T4 DNA ligase revealed by in vitro selection, *Nucleic Acids Res* 21, 2287-2291 (1993).
25. Harder, P.; Grunze, M.; Dahint, R.; Whitesides, G. M. and Laibinis, P. E., Molecular Conformation in Oligo(ethylene glycol)-Terminated Self-Assembled Monolayers on Gold and Silver Surfaces Determines Their Ability To Resist Protein Adsorption, *J. Phys. Chem. B* 1998,102, 426-436.
26. D. Herschlag (1991), Implications of ribozyme kinetics for targeting the cleavage of specific RNA molecules in vivo, *Proc. Nat. Acad. Sci. (USA)*, 88, pp. 6921-6925.
27. Jhaveri, S., Kirby, R., Conrad, R., Magion, E.J., Glick, G., Ellington, A.D. (1999) Signaling aptamers. Accepted to JACS.
28. Jonoska, N., S. Karl, M. Saito: Creating 3-Dimensional Graph Structures With DNA in DNA based computer III (Editors: H. Rubin, D. Wood) DIMACS series in Discrete Math. and Theoretical Comp. Sci. vol 48 (1999) 123-136.
29. Jonoska, N., S. Karl, M. Saito: Three dimensional DNA structures in computing (to appear in BioSystems).
30. Jonoska, N., 3D DNA patterns and Computing (to appear) collection of papers on Patterns in Biology edited by M. Gromov and A. Carbone (to be published by World Scientific).
31. Jonoska, N., S. Karl: Ligation Experiments in Computing with DNA, Proceedings of 1997 IEEE International Conference on Evolutionary Computation (ICEC'97), April 13-16, (1997) 261-265.
32. Jonoska, N., S. Karl: A molecular computation of the road coloring problem in DNA based computer II (Editors: L. Landwaber, E. Baum) DIMACS series in Discrete Math. and Theoretical Comp. Sci. vol 44 (1999) 87 - 96,
33. Jonoska, N., S. Karl, M. Saito: Graph structures in DNA computing in Computing with Bio-Molecules, theory and experiments, (editor Gh. Paun) Springer-Verlag (1998), 93-110.
34. Jonoska, N., Sofic Shifts with Synchronizing Presentations, *Theoretical Computer Science* vol. 158 1-2 (1996) 81-115.
35. Jonoska, N., Constants in factorial and prolongable languages, *Pure Math. Appl.* vol 7 1-2 (1996) 99-110.

36. LaBean, T. H., E. Winfree, J. H. Reif, Experimental Progress in Computation by Self-Assembly of DNA Tilings, 5th International Meeting on DNA Based Computers(DNA5), MIT, Cambridge, MA, (June, 1999). To appear in DIMACS Series in Discrete Mathematics and Theoretical Computer Science, ed. E. Winfree, to appear American Mathematical Society, 2000. <http://www.cs.duke.edu/~thl/tilings/labean.ps>
37. LaBean, T. H., Yan, H., Kopatsch, J., Liu, F., Winfree, E., Reif, J.H. and Seeman, N.C., The construction, analysis, ligation and self-assembly of DNA triple crossover complexes, *J. Am. Chem. Soc.* 122, 1848-1860 (2000). www.cs.duke.edu/~reif/paper/DNAtiling/tilings/JACS.pdf
38. Lagoudakis, M. G., T. H. LaBean, 2D DNA Self-Assembly for Satisfiability, 5th International Meeting on DNA Based Computers(DNA5), MIT, Cambridge, MA, (June, 1999). DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol.44, American Mathematical Society, ed. E. Winfree, (1999).
39. Lewis, H.R., and C.H. Papadimitriou, *Elements of the Theory of Computation*, Prentice-Hall, pages 296-300 and 345-348 (1981).
40. Li, X.J., X.P. Yang, J. Qi, and N.C. Seeman, Antiparallel DNA Double Crossover Molecules as Components for Nanoconstruction, *J. Am. Chem. Soc.*, 118, 6131-6140, (1996).
41. Liu, F., H. Wang and N.C. Seeman, Short Extensions to Sticky Ends for DNA Nanotechnology and DNA-Based Computation, *Nanobiology* 4, 257-262 (1999).
42. Liu, F., R. Sha and N.C. Seeman, Modifying the Surface Features of Two-Dimensional DNA Crystals, *J. Am. Chem. Soc.* 121, 917-922 (1999).
43. Liu, F., M.F. Bruist and N.C. Seeman, Parallel Helical Domains in DNA Branched Junctions Containing 5', 5' and 3', 3' Linkages, *Biochemistry* 38, 2832-2841 (1999).
44. Liu, Q., Wang, L., Frutos, A.G., Condon, A.E., Corn, R.M. and Smith, L.M., DNA computing on surfaces, *Nature* 403, 175-179 (2000).
45. Mao, W. Sun, Z. Shen and N.C. Seeman, A DNA Nanomechanical Device Based on the B-Z Transition, *Nature* 397, 144-146 (1999).
46. Mao, C., W. Sun, and N.C. Seeman, Construction of Borromean Rings from DNA, *Nature*, 386(6621), 137-138, (March,1997).
47. 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).
48. Mao, C., T.H. LaBean, J. H. Reif, and N.C. Seeman, An Algorithmic Self-Assembly, *Nature*, Sept 28, (2000). www.cs.duke.edu/~reif/paper/SELFASSEMBLE/AlgorithmicAssembly.pdf
49. Montemagno, C. D., and Bachand, G. D. Constructing nanomechanical devices powered by biomolecular motors. *Nanotechnology* 10: 225-331 (1999).
50. Montemagno, C. D., Bachand, G. D., Stelick, S. J., and Bachand, M. Constructing biological motor powered nanomechanical devices. Sixth Foresight Conference on Molecular Nanotechnology, (1998).
51. Moore, C. and J. M. Robson, Hard Tiling Problems with Simple Tiles, to appear, 2000.
52. Nielsen, P.E., Peptide Nucleic Acids, *Ann. Rev. Biophys. and Biophys. Chem.* 1995 14, 167-183.
53. Petty, M. C., M. R. Bryce and D. Bloor (Eds.), *An Introduction to Molecular Electronics*, Oxford University Press, New York (1995).
54. Reed, M. A., C. Zhou, C. J. Muller, T. P. Burgin and J. M. Tour, Conductance of a molecular junction, *Science*, Vol. 278, pages 252-254, October 10, 1997.

55. Reed, M. A. and J. M. Tour Computing with Molecules, *Scientific American*, June 2000. <http://www.scientificamerican.com/2000/0600issue/0600reed.html>
56. Reif, J.H., Local Parallel Biomolecular Computation, Third Annual DIMACS Workshop on DNA Based Computers, University of Pennsylvania, June 23-26, 1997. Published in *DNA Based Computers, III, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, Vol 48 (ed. H. Rubin), American Mathematical Society, 1999, p 217-254. Postscript versions of this paper and its figures are at <http://www.cs.duke.edu/~reif/paper/Assembly.ps> and [Assembly.fig.ps](http://www.cs.duke.edu/~reif/paper/Assembly.fig.ps)
57. Reif, J.H., Parallel Molecular Computation: Models and Simulations. *Proceedings: 7th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA'95)* Santa Barbara, CA, July 1995, pp. 213-223. Published in *Algorithmica*, special issue on Computational Biology, 25:142-176, 1999. www.cs.duke.edu/~reif/paper/Molecular.ps
58. Reif, J.H., *Synthesis of Parallel Algorithms*. 22 chapters, 1011 pages. Published by Morgan Kaufmann, Spring, 1993.
59. Reif, J.H., Paradigms for Biomolecular Computation, First International Conference on Unconventional Models of Computation, Auckland, New Zealand, January 1998. Published in *Unconventional Models of Computation*, edited by C.S. Calude, J. Casti, and M.J. Dinneen, Springer Publishers, 1998, pp 72-93. <http://www.cs.duke.edu/~reif/paper/paradigm.ps>
60. Reif, J.H., and T. H. LaBean, Computationally Inspired Biotechnologies: Improved DNA Synthesis and Associative Search Using Error-Correcting Codes and Vector-Quantization, Sixth International Meeting on DNA Based Computers (DNA6), DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Leiden, The Netherlands, (June, 2000) ed. A. Condon. To be published by the American Mathematical Society, 2000. www.cs.duke.edu/~reif/paper/Error-Restore/Error-Restore.ps
61. Reif, J.H. and Zheng Sun, Nano-Robotics Motion Planning and Its Applications in Nanotechnology and Biomolecular Computing, NSF Design & Manufacturing Grantees Conference, Jan 5-8, 1999. <http://www.cs.duke.edu/~reif/paper/NanoRobotics.html>
62. Robinson, R.M. Undecidability and Nonperiodicity for Tilings of the Plane, *Inventiones Mathematicae*, 12, 177-209, (1971).
63. Rothmund, P.W.K., Using lateral capillary forces to compute by self-assembly, *Proc. Nat. Acad. Sci. (USA)* 97, 984-989 (2000a).
64. Rothmund, P.W.K., and E. Winfree, The Program-Size Complexity of Self-Assembled Squares, *Symposium on Theory of Computing (STOC 2000)*, Portland, OR, (May, 2000b).
65. Roweis, S. E. Winfree. On the reduction of errors in DNA computation. *Journal of Computational Biology*, 6(1): 65-75, 1999.
66. Roweis, S. E. Winfree, R. Burgoyne, N. V. Chelyapov, M. F. Goodman, P. W. K. Rothmund, L. M. Adleman. A Sticker-Based Model for DNA Computation. *Journal of Computational Biology*, 5(4): 615-629, 1998
67. Seeman, N.C., Nucleic Acid Nanostructures and Topology. *Angewandte Chemie*. 110, 3408-3428 (1998); *Angewandte Chemie International Edition* 37, 3220-3238 (1998).
68. Seeman, N.C., DNA Engineering and its Application to Nanotechnology, *Trends in Biotech.* 17, 437-443 (1999).
69. Seeman, N.C. Nucleic Acid Junctions and Lattices, *J. Theor. Biol.*, 99, 237-247, (1982).

70. Seeman, N. C., J. Chen, S.M. Du, John E. Mueller, Yuwen Zhang, Tsu-Ju Fu, Yinli Wang, Hui Wang, Siwei Zhang, Synthetic DNA knots and catenanes, *New Jour. of Chemistry*, 17, 739-755, (1993).
71. Seeman, N. C., J.-H. Chen, N.R. Kallenbach, Gel electrophoretic analysis of DNA branched junctions, *Electrophoresis*, 10, 345-354, (1989).
72. Seeman, N. C., F. Liu, C. Mao, X. Yang, L.A. Wenzler, R. Sha, W. Sun, Z. Shen, X. Li, J. Qi, Y. Zhang, T. Fu, J.-H. Chen, and E. Winfree, Two Dimensions and Two States in DNA Nanotechnology, *Journal of Biomolecular Structure and Dynamics*, ISSN 0739-1102, Conversion 11, Issue 1, June, 1999.
73. Seeman, N. C., H. Wang, X. Yang, F. Liu, C. Mao, W. Sun, L.A. Wenzler, Z. Shen, R. Sha, H. Yan, M.H. Wong, P. Sa-Ardyen, B. Lui, H. Qiu, X. Li, J. Qi, S.M. Du, Y. Zhang, J.E. Mueller, T.-J. Fu, Y. Wang, and J. Chen, New Motifs in DNA nanotechnology, *Nanotechnology* 9, p 257-273 (1998).
74. Seeman, N. C., H. Wang, B. Liu, J. Qi, X. Li, X. Yang, F. Liu, W. Sun, Z. Shen, R. Sha, C. Mao, Y. Wang, S. Zhang, T.-J. Fu, S. Du, J. E. Mueller, Y. Zhang, and J. Chen, The Perils of Polynucleotides: The Experimental Gap Between the Design and Assembly of Unusual DNA Structures, *The 2nd Annual Workshop on DNA Based Computers*, American Mathematical Society, June 1996.
75. Seeman, N. C., Y. Zhang, and J. Chen, DNA nanoconstructions, *J. Vac. Sci. Technol.*, 12:4, 1895-1905, (1994).
76. Seeman, N. C., Y. Zhang, S. Du, H. Wang, J.E. Mueller, and J. Chen, The control of DNA structure and topology: An overview, *Mat. Res. Soc. Symp. Proc.*, 356, 57-66, (1994). H. Wang, Proving Theorems by Pattern Recognition, *Bell System Technical Journal*, 40, 1-141, (1961).
77. Sha, R., F. Liu, M.F. Bruist and N.C. Seeman, Parallel Helical Domains in DNA Branched Junctions Containing 5', 5' and 3', 3' Linkages, *Biochemistry* 38, 2832-2841 (1999).
78. Smith, L. M. , R. M. Corn, A. E. Condon, M. G. Lagally, A. G. Frutos, Q. Liu, and A. J. Thiel, A Surface-Based Approach to DNA Computation. *J. Computational Biology*, 5(2), 255-266. 1998.
79. Soong, R. K., Stelick, S. J., Bachand, G. D., and Montemagno, C. D. Evaluating adhesion strength of biological molecules to nanofabricated substrates. *Technical Proceedings of the Second Conference on Modeling and Simulation of Microsystems 99 Conference*; Puerto Rico (1999).
80. Sun, W., C. Mao, F. Liu and N.C. Seeman, Sequence Dependence of Branch Migratory Minima. *J. Mol. Biol.* 282, 59-70 (1998).
81. Wang, H., In *Proc. Symp. Math. Theory of Automata*, 23-26 (Polytechnic Press, New York, 1963).
82. Wang, Y., J.E. Mueller, B. Kemper, and N.C. Seeman, Assembly and characterization of five-arm and six-arm branched junctions, *Biochem.* 30, 5667-5674 (1991)
83. Winfree, E. Simulations of Computing by Self-Assembly. In *Proceedings of the Fourth Annual Meeting on DNA Based Computers*, held at the University of Pennsylvania, June 16-19, 1998.
84. Winfree, E., X. Yang, N.C. Seeman, Universal Computation via Self-assembly of DNA: Some Theory and Experiments, *2nd Annual DIMACS Meeting on DNA Based Computers*, Princeton, June, 1996.
85. Winfree, E., T. Eng, G. Rozenberg String tile models for DNA computing by self-assembly, unpublished notes, September 1998.
86. Winfree, E., Furong Liu, Lisa A. Wenzler, Nadrian C. Seeman (1998) Design and Self-Assembly of Two Dimensional DNA Crystals. *Nature* 394: 539-544, 1998.

87. Winfree, E., Xiaoping Yang, Nadrian C. Seeman. Universal Computation via Self-assembly of DNA: Some Theory and Experiments. In DNA Based Computers II: DIMACS Workshop, June 10-12, 1996 (Volume 44 in DIMACS). Laura F. Landweber and Eric B. Baum, editors. American Mathematical Society, 1998, pp. 191-213.
88. Winfree, E., On the Computational Power of DNA Annealing and Ligation. In DNA Based Computers: Proceedings of a DIMACS Workshop, April 4, 1995, Princeton University (Volume 27 in DIMACS). Richard J. Lipton and Eric B. Baum, editors. American Mathematical Society, 1996, pp. 187-198.
89. Winfree, E., F. Liu, L. A. Wenzler, and N.C. Seeman, Design and Self-Assembly of Two-Dimensional DNA Crystals, *Nature* 394, 539-544 (1998).
90. Winfree, E., 5th International Meeting on DNA Based Computers(DNA5), MIT, Cambridge, MA, (June, 1999). To appear in DIMACS Series in Discrete Mathematics and Theoretical Computer Science, ed. E. Winfree, (1999).
91. Xia, Y. and Whitesides, G. M., *Angew. Soft Lithography, Chem. Int. Ed. Engl.* 1998,37, 550-575.
92. Xia, Y. and Whitesides, G. M., *Soft Lithography, Annu. Rev. Mater. Sci.* 1998,28, 153-184. Yan, L.; Zhao, X.-M. and Whitesides, G. M. Patterning a Preformed, Reactive SAM Using Microcontact Printing, *J. Am. Chem. Soc.* 1998,120, 6179-6180.
93. Yang, X., L.A. Wenzler, J. Qi, X. Li and N.C. Seeman, Ligation of DNA Triangles Containing Double Crossover Molecules, *J. Am. Chem. Soc.* 120, 9779-9786 (1998).
94. Zhao, X.; Votruba, P.G.; Strother, T.C.; Ellison, M.D.; Smith, L.M. and Hamers, R.J. Formation of Organic Films on Si(111) Surfaces via Photochemical Reaction (in preparation), 1999.
95. Zhou, C. , et al., *Appl. Phys. Lett.* 71, 611 (1997).
96. Zhou, C. , Atomic and Molecular Wires, Ph.D. thesis, Yale University (1999).

A Space-Efficient Randomized DNA Algorithm for k -SAT

Kevin Chen^{*1} and Vijay Ramachandran^{**2}

¹ Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, Berkeley, CA 94720, USA, kevinc@eecs.berkeley.edu

² Department of Computer Science, Yale University, New Haven, CT 06520, USA, vijayr@cs.yale.edu

Abstract. We present a randomized DNA algorithm for k -SAT based on the classical algorithm of Paturi *et al.* [8]. For an n -variable, m -clause instance of k -SAT ($m > n$), our algorithm finds a satisfying assignment, assuming one exists, with probability $1 - e^{-\alpha}$, in worst-case time $O(k^2 mn)$ and space $O(2^{(1-\frac{1}{k})n + \log \alpha})$. This makes it the most space-efficient DNA k -SAT algorithm for $k > 3$ and $k < n/\log \alpha$ (i.e. the clause size is small compared to the number of variables). In addition, our algorithm is the first DNA algorithm to adapt techniques from the field of randomized classical algorithms.

1 Introduction

Ever since Lipton [4] showed how to solve Formula-SAT (of which k -SAT is a special case) with a DNA computer, a great deal of research within the field of DNA computing has been directed towards the design and implementation of SAT algorithms. Until Ogihara [6], however, all of these algorithms required that all 2^n possible solutions be generated initially. This implies an upper limit of about 50 variables on the size of the problem instance given the state of current biotechnology. In this paper, we follow Ogihara's strategy of generating putative solutions during the course of the algorithm, thereby reducing the number of strands required to be present at any one time during the computation and allowing us to solve larger instances of SAT.

1.1 Classical SAT Algorithms

The first complete SAT algorithms to be discovered were the Davis-Putnam algorithm (DP) [3] and the closely related Davis-Logemann-Loveland (DLL)

^{*} This work was done while at the Department of Computer Science, Princeton University, Princeton, NJ 08544, USA.

^{**} This work was done while at the Department of Mathematics, Princeton University, Princeton, NJ 08544, USA.

¹ A *complete* algorithm, as opposed to a *heuristic*, is guaranteed to find a solution if it exists. However, heuristics typically run much faster than complete algorithms.

algorithm [2]. Although both algorithms have been the subject of intense study in the last three decades, neither, surprisingly, has a proven worst-case time complexity. Nevertheless, their importance is such that virtually all of the most important complete SAT algorithms are more or less refined versions of DP and DLL. Monien and Speckenmeyer [5] gave the first improvement over DP and DLL, followed by a series of improved algorithms by Schiermeyer (1992), Kullman (1996), Schiermeyer (1996) and Paturi [8,7].

DP and DLL recursively branch on each variable by assigning the current variable the value “true” in one branch and “false” in the other. The computation may be viewed as a binary tree in which nodes represent the variables on which the algorithm branches and the edges represent assignments to the variables. At each branch point, the formula is simplified using a few simple rules. DP is based on the following two: 1) If the formula has a clause with only one literal, assign the variable of the literal the truth-value that satisfies the clause; 2) If some variable appears only positively, assign it the value “true”, and if it appears only negatively, assign it the value “false”. All the other complete SAT algorithms basically use a more complex set of simplification rules to achieve their improved time bounds. In particular, Paturi’s algorithm is essentially a randomized version of DP.

1.2 DNA SAT Algorithms

The first steps towards more space-efficient DNA SAT algorithms were taken by Ogihara [6]. His paper showed how to implement DP, DLL, and the Monien-Speckenmeyer algorithm [5] that has space complexity $O(2^{0.6942n})$ and time complexity $O(n \cdot \max\{m^2, n\})$, in its DNA form. Ogihara used the basic extract model of Lipton and Adleman plus the additional APPEND primitive. This extended model was first introduced by Boneh *et al.* in [1].

In addition, Ogihara gave an algorithm for 3-SAT with running time at most twice that of Lipton but with unproven space complexity. Computer experiments suggested a space complexity of about $2^{0.5n}$ and Yoshida and Suyama [9] later implemented a similar heuristic on a 4-variable, 10-clause 3-SAT instance.

2 Model of Computation

Following Ogihara [6], we use an extended version of the extract model that includes the APPEND primitive. In addition, for reasons to be clarified in the next section, we include the additional operations TO-SINGLE-STRANDED, that makes a double-stranded polymer single-stranded, TO-DOUBLE-STRANDED, that makes a single-stranded polymer double-stranded, and POUR, that divides the contents of a test tube into more than one test tube, thereby randomly splitting up the strands.

We assume our variables are x_1, x_2, \dots, x_n and that our SAT instance has m clauses.

2.1 Form of DNA Molecules

Our algorithm requires $2n + 3$ well-behaved sequences of DNA:

1. A header sequence, \mathbf{h} .
2. A separator sequence, \mathbf{s} .
3. A primer sequence, \mathbf{p} .
4. n “true” sequences, each denoted \mathbf{x}_i^T , representing the assignment “ $x_i = \text{true}$ ”.
5. n “false” sequences, each denoted \mathbf{x}_i^F , representing the assignment “ $x_i = \text{false}$ ”.

We will use the notation $\bar{\mathbf{a}}$ to mean the Watson-Crick complement of sequence \mathbf{a} .

The algorithm requires synthesis of $2n$ *assignment sequences* which are used to append variable assignments to solution strands. There is a true and false sequence for each variable x_i , denoted by $\mathbf{b}_i^{T/F}$, of the form shown in Fig. 1.

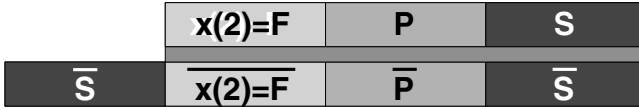


Fig. 1. Structure of \mathbf{b}_2^F , assignment sequence for “ $x_2 = \text{false}$ ”. Each box (\mathbf{s} , \mathbf{p} , $\bar{\mathbf{s}}$, and so on) represents a DNA subsequence discussed in Sect. 2.1. $\bar{\mathbf{s}}$ is the sticky end that anneals to the \mathbf{s} sticky end on the solution strand during an APPEND.

2.2 Operations

The allowable operations are now:

1. $\text{APPEND}(t, \{\mathbf{s}_1, \mathbf{s}_2, \dots, \mathbf{s}_k\})$: append to the end of each strand in tube t one of the subsequences $\mathbf{s}_1, \mathbf{s}_2, \dots, \mathbf{s}_k$. at random. This operation is a generalization of the standard APPEND introduced by [1]. We use this to append one variable assignment at a time. See Sect. 3.4 for an implementation of this operation.
2. $u \leftarrow \text{COMBINE}(t_1, t_2, \dots, t_k)$: combine the contents of tubes t_1 through t_k into a single tube u . Tubes t_1, \dots, t_n are left empty (unless, of course, $u = t_i$ for some $1 \leq i \leq k$).
3. $\text{DETECT}(t)$: select one strand at random from tube t , if any, and sequence it.
4. $u \leftarrow \text{EXTRACT}(t, \mathbf{s})$: extract from tube t all strands containing the subsequence \mathbf{s} and place them in tube u . We will normally extract on one of the sequences listed in Sect. 2.1.
5. $\{u_1, u_2, \dots, u_k\} \leftarrow \text{POUR}(t)$: pour out, or aliquot, the contents of t into k equal portions in test tubes u_1 through u_k . Tube t is left empty.

- 6. TO-DOUBLE-STRANDED(t): make each of the single-stranded molecules in tube t double-stranded except for a sticky end. See Sect. 3.4 for an implementation of this operation, also shown in Fig. 2.
- 7. TO-SINGLE-STRANDED(t): denature each double-stranded molecule in tube t and remove one strand, leaving the other as a single-stranded molecule in t . See Sect. 3.4 for an implementation of this operation.

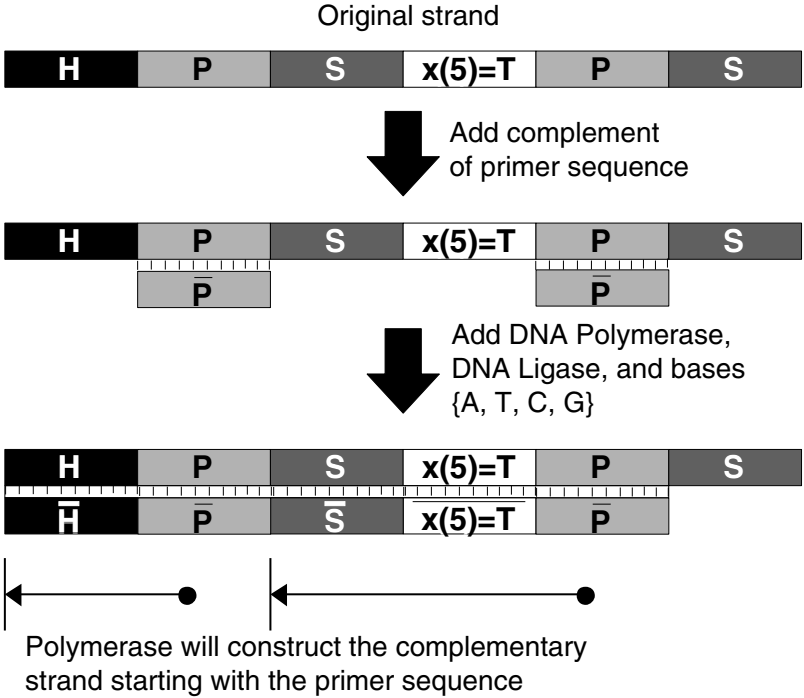


Fig. 2. Implementation of operation TO-DOUBLE-STRANDED.

The reason for the increased number of operations used in our model is that in our algorithm, the assembly of potential solution strands is very involved whereas this step in most other algorithms is relatively easy. However, all the biotechniques required to implement these operations are standard procedures used in other extract-based DNA algorithms.

Finally, we note that our measure of time complexity will be the number of extract steps in the computation, and the space complexity will be the number of strands in the system.

3 Algorithm

First we shall review the randomized classical k -SAT algorithm in [8] and [10] that we follow. We will then present and analyze our DNA implementation of it.

3.1 Notation and Definitions

x_1, x_2, \dots, x_n are our variables. The length of a *clause* $C = l_1 \vee l_2 \vee \dots \vee l_k$, or $|C|$, is simply k . Each $l_i \in C$ is a *literal*, which is either x_j or \bar{x}_j for some variable x_j . A *boolean formula* $F = \bigwedge_{i=1}^m C_i$ is a k -CNF formula if $\forall C \in F, |C| \leq k$.

3.2 The Algorithm Search

Given some large integer I and a k -CNF formula F , the algorithm in [8] (called **Search**) constructs I putative solutions and tests if they satisfy F . If no solution is found, the algorithm outputs “unsatisfiable.” The value of I depends on the error probability of **Search** and will be discussed at greater length later.

To construct one putative solution, at each iteration **Search** randomly selects some unassigned variable x_i and applies the unit clause rule from DP, implemented as follows. We say that a variable x_i is *forced* if it appears in some clause in which all other literals are “false” due to previous assignments. Clearly, if such a clause exists, F can be satisfied only by assigning x_i the value that makes that clause true, so that assignment is added to the solution. If no such clause exists, x_i is simply assigned a truth value at random. When all variables have been assigned, **Search** checks if the putative solution satisfies F .

A condensed version of **Search** is given below.

```

Begin Algorithm Search (formula  $F$ , positive integer  $I$ )
  Repeat  $I$  times
     $\pi$  = uniformly random permutation of  $\{1, 2, \dots, n\}$ 
    For  $i = 1$  to  $n$ 
      If  $x_{\pi(i)}$  is forced Then
        set  $x_{\pi(i)} =$  “true” or “false” as required
      Else
        set  $x_{\pi(i)} =$  “true” or “false” randomly
      End If
    End For
    If  $F$  is satisfied
      output assignment
      exit
    End If
  End Repeat
  output ‘Unsatisfiable’ /* No satisfying assignment found after  $I$  tries */
End Algorithm

```

3.3 Overview of DNA Implementation

To implement this algorithm in DNA, we construct all I potential solutions in parallel and then verify them all for correctness, in parallel, using Lipton's algorithm [4].

We begin with a tube containing I header sequences. In order to select a random unassigned variable, we assume that we have on hand n random permutations π_1 to π_n of the integers 1 to n . At the j^{th} iteration, we pour out the contents of the tube into n equal portions in tubes t_1 to t_n , and for each tube t_i we associate the variable $x_{\pi_j(i)}$. For each t_i , we would like to assign a value to $x_{\pi_j(i)}$. However, $x_{\pi_j(i)}$ may have already been assigned a value. To prevent a variable from being assigned a value twice, for each t_i we perform two extracts on the tube in parallel using the sequences for " $x_{\pi_j(i)} = \text{true}$ " and " $x_{\pi_j(i)} = \text{false}$ ". Having removed all strands for which the variable has been previously assigned from t_i , we add these strands to t_{i+1} and perform the respective extracts on it. Since the variable associated with t_{i+1} is picked at random according to π_j , we preserve the randomness in our choice of variable to be considered next. After performing the two extracts on t_n , we add the extracted strands to t_1 and repeat the process in its entirety once more. After two passes through all n tubes, each tube contains only strands in which the variable associated with the tube is unassigned.

To determine the assignment for a strand, we follow Ogihara's implementation of the unit clause rule [6]. Consider, for example, a tube t_i , and suppose its associated variable $x_{\pi_j(i)} = x_1$. Then for clause $C = x_1 \vee x_2 \vee x_3$, we need to extract on " $x_2 = \text{false}$ " and " $x_3 = \text{false}$ ". We perform similar extracts for all other clauses containing x_1 or $\overline{x_1}$. Formally, for each clause $C \in F$ containing $x_{\pi_j(i)}$ or $\overline{x_{\pi_j(i)}}$, we perform $k - 1$ extracts in series on t_i using the values of the $k - 1$ other variables in C that make their corresponding literals in C false. The extracted strands all are *forced* to have their value set to "true" because of this clause C . Forced strands in t_i will have the sequence representing the forced assignment appended to them. Strands not forced will have a random assignment for the variable appended to them.

All test tubes are then combined and the process, beginning with the random selection of an unassigned variable, is repeated until all strands have assignments for all variables. This occurs after n iterations. Finally, Lipton's algorithm [4] is used to select which of the constructed solutions in fact satisfies F .

3.4 Appending Truth Value Assignments

To assign x_i a truth value in a putative solution, we append an assignment sequence $\mathbf{b}_i^{\text{T/F}}$ (see Sect. 2.1) to the end of the solution strand. We begin with I copies of the header strand \mathbf{hps} , and each $\mathbf{b}_i^{\text{T/F}}$ that we append to our strands ends with the subsequence \mathbf{ps} . Therefore at each step where we perform APPEND, our solution strands end with the same subsequence \mathbf{s} . Since each $\mathbf{b}_i^{\text{T/F}}$ has a $\overline{\mathbf{s}}$ sticky end, it can anneal to that subsequence \mathbf{s} . However, our solution strands

contain multiple occurrences of \mathbf{s} , so we must first make them double-stranded except for the \mathbf{s} left exposed at the end.

For this purpose we have the operation TO-DOUBLE-STRANDED. We implement this by adding many copies of $\bar{\mathbf{p}}$ to t and then adding DNA polymerase. This enzyme should create the complement strand beginning with $\bar{\mathbf{p}}$ at the end of the strand, thus making the solution strands completely double-stranded except for the sticky end \mathbf{s} . (See Fig. 2) We are now able to add assignment sequences to the solution, knowing they will anneal to the proper location.

If we want to append a forced assignment to tube t , we can simply add many copies of the appropriate assignment sequence \mathbf{b}_i to t . For example, to assign “ $x_3 = \text{true}$ ” to all strands in t , we implement APPEND by adding \mathbf{b}_3^T to t .

To perform APPEND($t, \{\mathbf{b}_i^T, \mathbf{b}_i^F\}$), where we append “true” or “false” randomly to each strand, we simply add equal concentrations of both assignment sequences to t . If equal concentrations of \mathbf{b}_i^T and \mathbf{b}_i^F are added to tube t containing strands ending with \mathbf{s} , the probability that a true (or false) sequence actually anneals to a given strand should be $1/2$. Since our strands are all double-stranded except for the exposed sticky end \mathbf{s} or $\bar{\mathbf{s}}$, attraction between different parts of our molecules should not skew this probability. Thus each strand truly receives an assignment for x_i randomly.

We can now revert to single strands (used in the rest of the algorithm) by performing TO-SINGLE-STRANDED(t), which we implement as follows. First add DNA ligase to t . Then denature the strands in the tube to split each double-stranded molecule into single strands. Finally, perform EXTRACT($t, \bar{\mathbf{h}}$) to remove those strands not containing the header sequence $\bar{\mathbf{h}}$, thereby preserving only the original solution strands. These strands now all contain the newly appended assignment and end in \mathbf{ps} , ready for the next APPEND operation.

3.5 The Algorithm DNASearch

The pseudocode for our DNA implementation of Search follows. The notation defined in section 3.4 is used for DNA sequences throughout the algorithm. In addition, the sequence \mathbf{y}_j means the sequence representing an assignment that makes the literal l_j false.

```

Begin Algorithm DNASearch (formula  $F$ , integer  $s$ , positive integer  $I$ )
   $t_0 \leftarrow I$  copies of  $\mathbf{hps}$ 
  For  $c = 1$  to  $n$ 
     $\pi =$  random permutation of  $\{1, 2, \dots, n\}$ 
     $\{t_1, t_2, \dots, t_n\} \leftarrow \text{POUR}(t_0)$ 
    Repeat 2 times
      For  $i = 1$  to  $n$ 
         $u \leftarrow \text{EXTRACT}(t_i, \mathbf{x}_{\pi(i)}^T)$ 
         $u \leftarrow \text{EXTRACT}(t_i, \mathbf{x}_{\pi(i)}^F)$ 
         $t_{i+1} \leftarrow \text{COMBINE}(t_{i+1}, u)$  /* assume  $t_{n+1} = t_1$  */
      End For
    End Repeat
  For  $i = 1$  to  $n$  [in parallel]

```

```

For Each clause  $C \in F$  containing  $x_{\pi(i)}$  or  $\overline{x_{\pi(i)}}$ 
   $u \leftarrow \text{POUR}(t_i)$ 
  For Each literal  $l_j \in C$  where  $l_j \neq x_{\pi(i)}$  and  $l_j \neq \overline{x_{\pi(i)}}$ 
     $v \leftarrow \text{EXTRACT}(u, \mathbf{y}_j)$ 
     $t_i \leftarrow \text{COMBINE}(t_i, u)$ 
     $u \leftarrow \text{POUR}(v)$ 
  End For
  If  $x_{\pi(i)} \in C$  Then
     $z_i^{\mathbf{T}} \leftarrow \text{COMBINE}(z_i^{\mathbf{T}}, u)$ 
  Else  $/* \overline{x_{\pi(i)}} \in C */$ 
     $z_i^{\mathbf{F}} \leftarrow \text{COMBINE}(z_i^{\mathbf{F}}, u)$ 
  End If
End For
APPEND( $z_i^{\mathbf{T}}, \mathbf{b}_{\pi(i)}^{\mathbf{T}}$ )
APPEND( $z_i^{\mathbf{F}}, \mathbf{b}_{\pi(i)}^{\mathbf{F}}$ )
TO-DOUBLE-STRANDED( $t_i$ )
APPEND( $t_i, \{\mathbf{b}_{\pi(i)}^{\mathbf{T}}, \mathbf{b}_{\pi(i)}^{\mathbf{F}}\}$ )
End For
 $t_0 \leftarrow \text{COMBINE}(t_1, \dots, t_n, z_1^{\mathbf{T}}, \dots, z_n^{\mathbf{T}}, z_1^{\mathbf{F}}, \dots, z_n^{\mathbf{F}})$ 
TO-SINGLE-STRANDED( $t_0$ )
End For
Lipton( $F, t_0$ )  $/*$  verify satisfiability for strands in  $t_0$   $*/$ 
End Algorithm

```

3.6 Analysis of DNASearch

DNASearch runs a loop n times to construct putative solutions and then runs Lipton's algorithm to check the solutions. Lipton's algorithm runs in time $O(km)$ [4].

In the loop, we first POUR our strands into n different test tubes, which can be done in linear time. We must then perform $4n$ extracts (or $2n$ extracts, if “false” and “true” are done in parallel) in series to prevent multiple assignments of the same variable, taking $O(n)$ time.

Next, we check for *forced* variables. Consider a clause C containing k literals. For each literal, we need to perform, at some point, $k - 1$ extracts on the other literals in its clause, or $k(k - 1)$ extracts in all. Since there are m clauses, the clause checking process takes, in total, $m \cdot k \cdot (k - 1) = O(k^2m)$ time.

Finally, the remaining steps (such as APPEND) all take $O(1)$ time. Therefore each loop takes $O(n) + O(k^2m)$ time, and since the loop is executed n times, the total running time of DNASearch is $O(n^2 + k^2mn)$. Assuming $m > n$, the running time is simply $O(k^2mn)$.

To discuss the space complexity of DNASearch we must first discuss the error probability of the algorithm because it is randomized. Search and DNASearch will always return “unsatisfiable” if F indeed cannot be satisfied. However, there is a chance that the algorithm will report F is unsatisfiable when it can be satisfied. This occurs when an actual solution for F is not generated as one of the I strands produced in the algorithm.

Paturi *et al.* prove in [8] that the probability this will occur for a given I is at most

$$e^{-I \cdot 2^{-(1-\frac{1}{k})n}}.$$

(For a full explanation of the proof, see [10].)

Therefore, if $I = \alpha 2^{(1-\frac{1}{k})n} = 2^{(1-\frac{1}{k})n + \log \alpha}$, the algorithm has error probability $O(e^{-\alpha})$. Note that increasing the error probability decreases I . Since the number of strands, or the space, required by DNASearch is I , this proves that the space complexity of the algorithm with error probability $O(e^{-\alpha})$ is indeed $O(2^{(1-\frac{1}{k})n + \log \alpha})$.

4 Discussion

Our paper opens up many opportunities for further work. More research needs to be done on harnessing the tools of randomized classical algorithms and the inherent randomness of molecular computation together in designing new DNA algorithms. Furthermore, it is an interesting open question whether or not there are general principles that govern the “blow-up” in time complexity when a classical randomized algorithm is transformed into a DNA algorithm. Clearly this depends on the model of computation of the DNA algorithm. For example, in our model, the sources of randomness are the POUR and APPEND operations, but we could conceivably use other operations, possibly yielding a different running time.

In a later paper [7], Paturi *et al.* introduced a preprocessing step for Search that improves the overall running time to $O(2^{0.446n})$ for 3-SAT. This step, called Resolve, adds clauses to the formula F to make it more probable that a solution will be found quickly. (In other words, the Resolve step reduces I .) However, incorporating Resolve in a DNA algorithm is difficult because it adds $2^{o(n)}$ clauses to F . Adding this exponential time preprocessing step to our DNA algorithm would produce an exponential running time and exponential space algorithm. It may be possible to implement Resolve in parallel in DNA, although further research is necessary to determine the feasibility of this technique.

Acknowledgements. We would like to thank Erik Winfree and Richard Lipton for their valuable comments throughout all stages of this work, and Krista Dobi for her insights into laboratory techniques.

References

1. D. Boneh, C. Dunworth, R. Lipton, and J. Sgall. On the computational power of DNA. *Discrete Applied Mathematics: Special Issue on Computational Molecular Biology*, 71:79–94, 1996.
2. M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5:394–397, 1962.

3. M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, 1960.
4. R. Lipton. Using DNA to solve NP-complete problems. *Science*, 268:542–545, April 1995.
5. B. Monien and E. Speckenmeyer. Solving satisfiability in less than 2^n steps. *Discrete Applied Mathematics*, 10:287–295, 1985.
6. M. Ogihara. Breadth first search 3-SAT algorithms for DNA computers. Technical Report TR 629, University of Rochester, Department of Computer Science, Rochester, NY, July 1996.
7. R. Paturi, P. Pudlák, M. Saks, and F. Zane. An improved exponential-time algorithm for k -SAT. In *39th Annual Symposium on Foundations of Computer Science*, pages 628–637, Palo Alto, California, 8–11 Nov. 1998. IEEE.
8. R. Paturi, P. Pudlák, and F. Zane. Satisfiability coding lemma. In *38th Annual Symposium on Foundations of Computer Science*, pages 566–574, Miami Beach, Florida, 20–22 Oct. 1997. IEEE.
9. H. Yoshida and A. Suyama. Solution to 3-SAT by breadth-first search. In E. Winfree and D. Gifford, editors, *DNA Based Computers V*, Cambridge, Massachusetts, 14–15 June 1999. American Mathematical Society.
10. F. Zane. *Circuits, CNFs, and Satisfiability*. PhD thesis, University of California at San Diego, Department of Computer Science and Engineering, 1998.

A DNA-Based Random Walk Method for Solving k -SAT

Sergio Díaz^{1*}, Juan Luis Esteban^{1*}, and Mitsunori Ogihara^{2**}

¹ Dept. Llenguatges i sistemes informàtics, Universitat Politècnica de Catalunya
c/ Jordi Girona Salgado 1-3, 08023 Barcelona, Spain

{sdiaz,esteban}@lsi.upc.es

² Department of Computer Science, University of Rochester
Rochester, NY, USA.

ogihara@cs.rochester.edu

Abstract. This paper presents an implementation of a concurrent version of Schönning's algorithm for k -SAT in [Sch99]. It is shown that the algorithm can be implemented with space complexity $O((2 - \frac{2}{k})^n)$ and time complexity $O(kmn + n^3)$, where n is the number of variables and m the number of clauses. Besides, borrowing ideas from the above mentioned implementation, it is presented an implementation of resolution, a widely studied and used proof system, mainly in the fields of Proof Complexity and Automated Theorem Proving.

1 Introduction

Developing efficient DNA-based algorithms for solving NP-complete problems is one of the most important issues of DNA-based computation. The celebrated paper by Adleman [Adl94], and the subsequent generalization by Lipton [Lip95], explored the possibility of solving NP-complete problems using DNA. Both Adleman and Lipton addressed the issue of coping with exponentially growing amount of DNA used as the size of instances that can be solved with their algorithms is quite limited. It is crucial that the growth in the amount of DNA used is suppressed in order for DNA based computation to compete against silicon-based computers. Several proposals have been made in the past (see [ORS97, OR99] for a survey) to resolve the problem of reducing the amount of DNA to be used. In particular, the following three proposals seem important: (1) Bach *et al.* [BCGT96] proposed methods for solving such NP-complete problems as Clique and Independent Set, where solution candidates to the instance is generated by combining (in *all* possible combinations) solutions for its subproblems. (2) Ogihara [Ogi96] (see also [OR97]) proposed breadth-first search methods for solving SAT, where partial assignments to the instance given are gradually extended from the empty assignment towards the full assignments and during the

* Supported by MEC through grant PB98-0937-C04 (FRESCO project)

** Supported in part by the National Science Foundation Grants CCR-9701911, CCR-9725021 and INT-9726724.

extension process those partial assignments that already fail to satisfy at least one clause are eliminated. (3) Cai *et al.* [CCC97] (see also [CCC97,LWF]) proposed the surface-based DNA computation, where biochemical operations are performed on single DNA strands immobilized on a gold surface.

The purpose of this paper is to explore one more direction in the study of “space-efficient” DNA algorithms for solving NP-complete problems. We study DNA-based parallel execution of a probabilistic algorithm for SAT. More precisely, we study possible implementation of Schöning’s algorithm for the k -SAT problem [Sch99]. k -SAT is the problem of testing satisfiability of CNF formulas each of whose clauses consists of exactly k literals. Schöning’s algorithm is essentially sequential repetition of a $2n$ step random walk procedure. In the random walk procedure one starts with a random assignment and executes at most $2n$ times the following:

pick uniformly at random a clause that is unsatisfied currently, pick a literal from the selected clause uniformly at random, and flip the assignment to the literal to satisfy the clause.

In Schöning’s algorithm, this random walk procedure is repeated $F_k(n) = (2 - \frac{2}{k})^n$ times to achieve a constant success probability of finding a satisfying assignment for a satisfiable formula of n variables.

The reader will perhaps immediately notice an interesting question arising from this Schöning’s result. Can we parallelize the sequential $F_k(n)$ repetition of the random walk procedure on DNA? Namely, is it possible to design a DNA-based algorithm where one starts with $F_k(n)$ randomly selected initial assignments and performs the $2n$ step random walk procedure concurrently on the pool of DNA. If that is possible, then we could reduce the space requirement for solving k -SAT from 2^n to $F_k(n) = (2 - \frac{2}{k})^n$, and that would increase the size of the largest formulas that could be handled by DNA based computation by a multiplicative factor of $\frac{k}{k-1}$. In particular, in the case when $k = 3$, the size would be $\frac{3}{2}$ of what it is with Lipton’s method.

We present in this paper one possible implementation of the algorithm. The paper is organized as follows. In Section 2 we describe the aforementioned DNA-based implementation. In Section 3 we present one small application of the algorithm we develop — resolution. In Section 4 we discuss open issues.

2 DNA-Implementation of Schöning’s Algorithm

2.1 The Computation Model

We use the test-tube model in which DNA strands are dissolved in water. Following Adleman [Adl94] and Lipton [Lip95], we permit the following operations on DNA:

1. **merge** — mixing the contents of two test tubes into one,
2. **synthesize** — chemically synthesizing specific patterns of DNA,
3. **anneal** — making DNA strands form double strands without elongation,

4. **append** — appending a given pattern to each single DNA strand (accomplished by annealing with DNA ligase),
5. **denature** — pulling away double-stranded DNA into single-stranded DNA,
6. **detect** — testing whether a given test tube contains a DNA strand,
7. **length** — separating DNA strands according to their base length,
8. **degenerate** — destroying single-stranded DNA into single DNA molecules (accomplished by the use of exonuclease),
9. **cut** — using restriction enzymes to cut DNA strands at specific restriction sites,
10. **separate** — separating DNA strands based on their pattern.

These are all well-studied biochemical operations (see [SFM89] or for a quick overview of them see [ORS97]). We note that recent technical advances have made the length technique robust and rapid [ROMJ97].

2.2 Schöning's Algorithm

Figure 1 describes Schöning's algorithm [Sch99].

Given a k -CNF formula φ of n variables, repeat the following $F_k(n) = (2(1 - \frac{1}{k}))^n$ times.

Step 1: Guess an initial assignment $a \in \{0, 1\}^n$ uniformly at random.

Step 2: Repeat $2n$ times:

- (a) If $\varphi(a) = 1$ (the current assignment satisfies the formula), then accept and halt.
- (b) Otherwise, from the set of all clauses that are not satisfied by the current assignment, pick one, C , uniformly at random.
- (c) Out of the k literals in C pick one, y , uniformly at random.
- (d) Flip the assignment to the variable in y .

Fig. 1. The Sequential Random Walk Procedure in Schöning's algorithm

Schöning shows that if the formula is satisfiable, with high probability one finds a satisfying assignment for φ and if the formula is unsatisfiable one never finds that φ is satisfiable.

2.3 Concurrent Version of Schöning's Algorithm

The body of the main loop of Schöning's algorithm is sequential repetition of the $2n$ step random walk procedure. Using massive parallelism of DNA computation one may perform individual $F_k(n)$ runs of the main body concurrently, namely, by executing the $2n$ step random walks for $F_k(n)$ independent assignments. If the assignments that are manipulated are encoded as a unique DNA strand,

such modification will yield a DNA-based algorithm requiring $O(F_k(n))$, if there is no worry of losing strands. So consider the following concurrent version of Schöning's algorithm (see Figure 2).

Step 1: Pick $F_k(n)$ random assignments as the initial search space.

Step 2: Repeat the following $2n$ times.

- (a) Test whether there is a satisfying assignment in the current pool. If so, assert that the input formula is satisfiable and halt.
- (b) Concurrently for each assignment a , select uniformly at random a clause C yet to be satisfied, select uniformly at random a literal $y \in C$, and flip the bit of y in a .

Fig. 2. Concurrent Version of Schöning's Algorithm

2.4 Encoding Scheme

In order to implement the concurrent version of the algorithm, we employ a specific DNA encoding scheme. Let $\varphi = C_1 \wedge \cdots \wedge C_m$ be a k -SAT formula over some n variables, whose satisfiability we are testing. Let Λ denote the set of all literals, i.e., $\Lambda = \{x_1, \dots, x_n\} \cup \{\bar{x}_1, \dots, \bar{x}_n\}$. An alternative way of expressing a literal is to use an equation $x_i \equiv b$, $1 \leq i \leq n$, $b \in \{0, 1\}$, where $x_i \equiv 0$ is for the negative literal \bar{x}_i and $x_i \equiv 1$ is for the positive literal x_i . We assign indices 1 to $2n$ to these literals, by letting the equation $x_i \equiv b$ correspond to the index $2i + b - 1$. For each i , $1 \leq i \leq 2n$, λ_i denotes the literal with index i . Each single-stranded DNA appearing during computation is the concatenation of short stretches of single strands selected from the set E consisting of:

1. *The Headers* $[H, +]$, $[H, -]$;
2. *The Bit Setters* $[B, i, +]$ and $[B, i, -]$, where $1 \leq i \leq 2n$;
3. *The Failed Literal Specifiers* $[F, i, j]$, where $1 \leq i \leq n$ and $1 \leq j \leq 2n$;
4. *The Bit Flip Locators* $[C, i]$, where $1 \leq i \leq n$.

We assume that the above strings are of equal length and the unique length of them is an even number, say L . The $+$ and $-$ signs used to indicated the parity of the round number in the random walks to be executed. The header strands, $[H, +]$ and $[H, -]$, are used as the header, in the sense that all legitimate encodings of assignments begin with one of the two headers. The bit setters are used to encode assignments to individual variables; namely, for each i , $1 \leq i \leq n$, $[B, i, +]$ and $[B, i, -]$ both encode that the literal for the $\lceil i/2 \rceil$ th variable is λ_i . For each legitimate truth assignment and for each variable i , $1 \leq i \leq n$, at most one of $\{x_i, \bar{x}_i\}$ can be on the list. The failed literal specifiers are used when we list for each assignment all the literals that it fails to satisfy. The strand $[F, i, j]$ being appended to an assignment indicates that λ_j is the i th smallest element in the set of all the literals that appear in at least one clause that the assignment

fails to satisfy. The bit flip locators are used to specify the literal for which a bit flip will be performed. One of these locators will be appended to each non-satisfying assignment immediately after the list of dissatisfied literals has been appended. The pattern $[C, i]$ being appended to an assignment indicates that the change will occur for the i th literal in the list appended.

To perform the operation **cut** we select a restriction enzyme \mathcal{R} that recognizes single-stranded pattern $\sigma\tau$ and cuts between σ and τ . We embed these patterns σ and τ into some of the strands.

- $[H, +]$ and $[H, -]$ start with τ ,
- for each i , $1 \leq i \leq 2n$, and $S \in \{+, -\}$, $[B, i, S]$ ends with σ but does not start with τ ,
- for each i , $1 \leq i \leq n$, and each j , $1 \leq j \leq 2n$, $[F, i, j]$ starts with τ and ends with σ ,
- for each i , $1 \leq i \leq n$, $[C, i]$ starts with τ and ends with σ , and
- neither σ nor τ appear in any other place.

For two basic components u and v in E , by the *linker* of $u \cdot v$, we mean the strand that is complementary antiparallel to the strand constructed by appending the 5'-end half of v to the 3'-end half of u . For simplicity we write $lnk(u, v)$ to denote the linker of $u \cdot v$.

These basic components are connected into a truth assignment in an obvious way: it is the header followed by assignments to x_1, x_2, \dots, x_n . We require these substrings should have a common parity value.

2.5 Implementing Step 1: Generation of the Initial Search Space

Random generation of $F_k(n)$ initial assignments is done by pouring in a test tube $F_k(n)$ copies of each of the following parts and let the strands anneal with DNA ligase:

1. *Header* $[H, +]$;
2. *Bit Setters* for each i , $1 \leq i \leq 2n$, $[B, i, +]$,
3. *Linkers*
 - a) $lnk([H, +], [B, 1, +])$, $lnk([H, +], [B, 2, +])$; and
 - b) for each i , $1 \leq i \leq n - 1$, $lnk([B, 2i - 1, +], [B, 2i + 1, +])$, $lnk([B, 2i - 1, +], [B, 2i + 2, +])$,
 $lnk([B, 2i, +], [B, 2i + 1, +])$, and $lnk([B, 2i, +], [B, 2i + 2, +])$.

Note that the linkers have effect of appending an assignment to x_0 to the header and of appending for each i , $1 \leq i \leq n - 1$, an assignment to x_{i+1} to an assignment to x_i . Since the header and bit selectors are of base length L all the legitimate strands generated will be of base length at most $(n + 1)L$ and only the legitimate strands of base length $(n + 1)L$ will encode assignments. Since the number of duplicates is $F_k(n)$ for each substring, if all the header strands are used, then $F_k(n)$ many strands of base length $(n + 1)L$ will be generated. We have only to extract strands of base length $(n + 1)L$. There are

various kinds of uncertainty involved in this process. First, formation of double strands arrives at an equilibrium. Second, strands may be lost during **merge** and **length** operations. Third, the yield of the **synthesis** operation is not perfectly controllable. Since the algorithm we are implementing is probabilistic, we can accommodate the uncertainty by simply increasing the amount of the strands to be put in by a certain amount.

2.6 Implementing Step 2a: Testing the Existence of Satisfying Assignment in the Pool

Although it is possible to implement this test as an independent step, in order to reduce the total number of biochemical operations in the implementation we will incorporate this step into the implementation of Step 2b described below.

2.7 Implementing Step 2b: Executing Concurrent Random Walk

As we noted before we use two time stamps, $+$ and $-$. Whenever we are about to execute Step 2 (combined Steps 2a and 2b) all the assignments in the test tube have a unique time stamp. From these assignments we will generate new ones by bit flipping, and the new ones will have the opposite time stamp. Let S be the time stamp that is unique to the current assignments and S' be the opposite time stamp. Let T denote the test tube containing all the current assignments.

The implementation of the random walk step is divided into the following three phases:

Phase 1 For each i , $1 \leq i \leq 2n$, select into the test tube Q_i assignments whose are to be modified to satisfy λ_i .

Phase 2 For each i , $1 \leq i \leq 2n$, from each assignment in Q_i create its copy with the new time stamp and with the intended modification to the assignment to λ_i .

Phase 3 Merge all the new assignments into one test tube. Turn that test tube into T .

The last phase will be executed by a sequence of **merge** operation, where $2n - 1$ test tubes will be mixed into another test tube.

Phase 1: Distributing assignments to test tubes. First we explain the idea. Let α be a non-satisfying assignment in the current pool. Suppose α fails to satisfy precisely h clauses, C_{t_1}, \dots, C_{t_h} , $t_1 < \dots < t_h$, and let $\lambda_{i_1}, \dots, \lambda_{i_d}$ be the enumeration of all the literals appearing in at least one of these clauses. We append $[F, 1, i_1], \dots, [F, d, i_d]$ to α in this order then append randomly one of $[C, i], 1 \leq i \leq d$ to it. Suppose $[C, r]$ be the one that is selected. Then combining $[C, r]$ and $[F, r, i_r]$ will specify that λ_{i_r} is the bit selection to be flipped. For that matter, for each i , $0 \leq i \leq n$, and j , $0 \leq j \leq i$, define $T_{i,j}$ to be the collection of all assignments α such that:

- the number of literals in $\{\lambda_1, \dots, \lambda_{2i}\}$ that appear in at least one clause that α fails to satisfy is precisely j .

Naturally $T_{0,0}$ is the current pool of assignments.

By dynamic programming we can compute all $T_{i,j}$, $0 \leq i \leq n$, $0 \leq j \leq i$ based upon the following recurrence:

Dynamic Programming Recurrence for $i = 1 \dots n$, for $j = 0 \dots i$, put all the assignments in $T_{i-1,j}$ that fail to satisfy at least one clause in which either λ_{2i-1} or λ_{2i} appears into $T_{i,j+1}$ and put the rest into $T_{i,j}$.

Obviously the current pool contains a satisfying assignment if and only if $T_{n,0}$ is empty.

Now the actual implementation is described in Figure 3.

- For $i = 1 \dots n$, do the following:
 - Concurrently for $j = 0, \dots, i$, prepare $T_{i,j}$ as an empty test tube.
 - Concurrently for $j = 1, \dots, i$, extract from $T_{i-1,j-1}$ into V_j all the assignments that fail to satisfy at least one clause in which λ_{2i-1} appears, and extract from the rest of $T_{i-1,j-1}$ into V'_j all the assignments that fail to satisfy at least one clause in which λ_{2i} appears; then merge the remainder into $T_{i,j-1}$.
 - Concurrently for $i = 1, \dots, j$, append $[F, j, 2i - 1]$ to all the strands in V_j and $[F, j, 2i]$ to all the strands in V'_j , then merge these into $T_{i,j}$.
- Test whether $T_{n,0}$ contains an assignment. If so, assert that the formula is satisfiable.
- Concurrently for $j = 1, \dots, n$, anneal with DNA ligase after merging $F_k(n)$ copies of each of the following in $T_{n,j}$:
 - $[C, l]$, $1 \leq l \leq j$, and the linkers $lnk([F, j, t], [C, l])$, $1 \leq t \leq 2n$, $1 \leq l \leq j$.
- Merge $T_{n,1}, \dots, T_{n,n}$ into a test tube U . By **separate**, divide U into test tubes U'_{ij} , $1 \leq i \leq n$, $1 \leq j \leq 2n$, where U'_{ij} consists of all strands in U with both $[C, i]$ and $[F, i, j]$ on them.
- For each j , $1 \leq j \leq 2n$, construct a test tube Q_j by merging all the test tubes U'_{1j}, \dots, U'_{nj} .
- For $u \in A$, remove the appended strands used for classification by using the enzyme \mathcal{R} and then by extracting with the **length** operation all the strands of base length $(n + 1)L$.

Fig. 3. Classification Algorithm

Note that Line 15 can be implemented by repeating k times **separate** steps as was used in Lipton's method for CNF-SAT (Lip95).

Phase 2: Flipping a bit. Now we will flip bits in tubes Q_r for $1 \leq r \leq 2n$. Let W be the test tube containing all the assignments for which we chose to flip a bit so that λ_r is satisfied. Let t and b such that λ_r is representing $x_t \equiv b$. That means that before the flipping the assignments satisfy $x_t \equiv 1 - b$, so if r is odd

then λ_{r-1} is satisfied and if r is even then λ_{r+1} is satisfied. The current time stamp, say S must be changed by the opposite, say S' . Figure 4 describes the algorithm.

1. Append $[H, S']$ to all the strands in W .
2. For $i = 1, \dots, n$, do the following
 - If $i \neq t$, then do the following:
 - a) For each $b \in \{0, 1\}$, extract from W into W_b all the strands with $[B, 2i - 1 + b, S]$, and append $[B, 2i - 1 + b, S']$ to all the strands in W_b ,
 - b) Merge W_0 and W_1 into W .
 - Otherwise, append $[B, r, S']$ to all the strands in W .
3. Cut the strands using the enzyme \mathcal{R} . Extract the strands with $[H, S']$.

Fig. 4. Bit Flipping Algorithm

Note that after the loop there are no well formed assignments. All the strands have double the size of a well formed assignment. The **cut** operation separates the old assignment from the new assignment. The strands with $[H, S']$ correspond to the new assignments. It is important to remark that the length of the strands is always $O(n)$.

Comment

- Add to the test tube many copies of the complement of each strand in E of time stamp S' and let the strands anneal (without DNA ligase).
- **Degenerate** to remove single stranded parts of partially double strands.
- **Denature** and then by the **length** operation, extract all the strands of base length $(n + 1)L$.

2.8 Running Time Analysis

Let us analyze the running time of the algorithm. The initial generation can be done in $O(1)$ steps. For each distribution phase, the number of concurrent **separate** steps required for producing the test tubes T_{ij} , is proportional to the total number of literals, so it is $O(km)$. Reclassification of the assignments based on the labels generated is proportional to $O(n^2)$. As there are $2n$ random walk steps the total running time is $O(kmn + n^3)$.

Theorem 1. *The concurrent version of Schönig's algorithm can be implemented with $O(kmn + n^3)$ steps.*

We need to worry about losing assignments during the execution of the *entire random walk steps*. Suppose the reduction rate is θ , $0 < \theta < 1$, for one step of random walk; i.e., after execution of a single step of concurrent random walk phase, the amount of assignments in the test tube will be decreased by a fraction of θ . (We can assume that θ is small.) In order to maintain at least $F_k(n)$ assignments at the end, we need to start with $F'_k(n) = F_k(n) \cdot (1 - \theta)^{-2n}$ assignments at the beginning. This changes the space complexity of the problem to $F'_k(n)$.

Theorem 2. *The concurrent version of Schönning's algorithm can be made to run in time $O(kmn + n^3)$ steps and in space $O(F'_k(n))$.*

3 DNA-Implementation of Resolution

3.1 Resolution

Resolution is one of the most widely used proof systems in computer applications, in particular, Automated Theorem Proving, and is based upon the resolution algorithm of Robinson [Rob65]. Resolution is a refutation system for negations of tautologies, expressed in CNF. This system has only one rule of inference:

$$\frac{C \vee \{x\} \quad D \vee \{\neg x\}}{C \vee D},$$

where $C \vee D$ is called the *resolvent*. In a resolution proof we simply apply the resolution rule starting from the initial clauses until the empty clause is obtained.

In order to speed up the time of finding refutations, some restrictions of the resolution rule have been proposed. Those restrictions are of course solid, and most of them complete. Most of these restriction have been proved to be *less efficient* than (unrestricted) resolution.

3.2 Implementation of Concurrent Resolution on a Variable

We will show that it is possible to implement with DNA concurrent resolution on one specific variable; i.e. given a pool of clauses, it is possible to do the following:

1. Search (based on the indices given to the variables) for a variable x for which there exist at least one clause containing x and at least one containing \bar{x} . If there is no such variable the formula is satisfiable.
2. Concurrently and randomly combine one clause with x and another with \bar{x} to perform resolution.
3. If the empty clause is generated then the formula is unsatisfiable. Otherwise return to the first step.

Note that each time we execute the above successfully, the number of variables appearing either positively or negatively in any of the clauses will decrease by at least one. So repeating the above procedure n times, for a formula of n variables, will be sufficient.

In order to implement the above procedure, we use the parity time stamp as we did in the previous section. The coding of each literal will be redundant in the sense that we expand the selection for each variable from two to three, namely, positive, negative, and not-existent. For a time stamp $S \in \{+, -\}$, we will use $[x_i, S]$, $[\bar{x}_i, S]$, $[\perp_i, S]$ to denote the three choices, respectively. For example for a clause $(x_1 \vee \bar{x}_3)$ for a formula over $\{x_1, \dots, x_4\}$, the encoding will look like $[x_1, S][\perp_2, S][\bar{x}_1, S][\perp_4, S]$. As in the previous algorithm, each assignment has a header $[H, S]$.

3.3 Selecting Any Variable

In order to test whether x_i can be used for resolution. Extract into T_1 all the strands with $[x, S]$ and into T_2 all the strands with $[\bar{x}, S]$. If both T_1 and T_0 are nonempty, we will use x_i . If exactly one of them is nonempty, we will throw away the empty one, since the clauses in it are all satisfiable.

3.4 Performing a Resolution Steps in Parallel

Once we have a suitable variable, say x_i , we merge T_0 and T_1 to make T' . We first connect the strands in T_0 after those in T_1 . For that matter, we append a special strand $[J, 1]$ after all the strands in T_0 and attach a special strand $[J, 2]$ before all the strands in T_1 . By linking $[J, 1]$ and $[J, 2]$, we join the strands in T_0 and T_1 in an arbitrary combination. Then we perform concurrent resolution step by sequence-based separation by following a similar method that we discussed in the previous section. After appending a new header $[H, S']$, for each j , $1 \leq j \leq n$, in this order, we do the following:

- If $j \neq i$, then
 - if the strand has both $[x_j, S]$ and $[\bar{x}_j, S]$, then throw the strand away;
 - if the strand has both $[x_j, S]$ and $[\perp_j, S]$, then append $[x_j, S']$;
 - if the strand has both $[\bar{x}_j, S]$ and $[\perp_j, S]$, then append $[\bar{x}_j, S']$;
 - if the strand has only $[\perp_j, S]$, then append $[\perp_j, S']$
- if $j = i$, then append $[\perp_i, S']$.

Then we anneal the complements of the new part, degenerate, and then take only length $(n + 1)L$ strands.

To check whether we have found the empty clause, we just check if there is any strand formed solely by the empty literals.

3.5 General Resolution

Now it is easy to understand how to perform unrestricted resolution. We connect the strands of tube T and get rid of those of incorrect length. In test tube T_x we will place all the strands with x and \bar{x} . Each variable will have its own test tube. Each test tube must be checked to throw away any clause that contains both any literal y and \bar{y} other than the attached to the tube. In the previous subsection we have explained how to perform the resolution step. Once done we throw the contents of all the tubes into T and check whether the empty clause is generated.

4 Conclusion

In this paper we presented an implementation of Schönning's method on DNA. We did *not* show, however, the optimality of the implementation, not even feasibility with the current technology.

In Theorem 2 we argue that Schönning's algorithm can be implemented in time $O(kmn + n^3)$ and in space $F'_k(n)$. In a more rigorous analysis the rate at

which the strand decreases in one random walk step would be in the form of $(1 - \xi)^{\Omega(n^2 + km)}$ as $\Omega(n^2 + km)$ operations are involved in one random walk step. The crucial issue here is to identify the best rate possible for the value Θ or for this value ξ . The rigorous estimate of the values will reveal the degree of feasibility of this implementation. Also one should explore more time efficient methods for implementing the algorithm.

Note that, in order for the restricted version or for the general version to be successful, one needs to start with a large number of copies of the literals. It is not clear to us how many copies are sufficient.

References

- [Adl94] L. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266:1021–1024, 1994.
- [BCGT96] E. Bach, A. Condon, E. Glaser, and C. Tanguay. DNA models and algorithms for NP-complete problems. In *Proceedings of 11th Conference on Computational Complexity*, pages 290–299. IEEE Computer Society Press, Los Alamitos, CA, 1996.
- [CCC97] W. Cai, A. Condon, R. Corn, E. Glaser, Z. Fei, T. Frutos, Z. Guo, M. Lagally, Q. Liu, L. Smith, and A. Thiel. The power of surface-based DNA computation. In *Proceedings of 1st International Conference on Computational Molecular Biology*, pages 67–74. ACM Press, 1997.
- [Lip95] R. Lipton. DNA solutions of hard computational problems. *Science*, 268:542–545, 1995.
- [LWF] Q. Liu, L. Wang, A. G. Frutos, R. M. Corn, and L. M. Smith. DNA computing on surfaces. *Nature*, 403:175–178, 2000. January, 13.
- [Ogi96] M. Oghihara. Breadth first search 3SAT algorithms for DNA computers. Technical Report TR 629, Department of Computer Science, University of Rochester, Rochester, NY, July 1996.
- [OR97] M. Oghihara and A. Ray. DNA-based parallel computation by counting. In H. Rubin and D. H. Wood, editors, *DNA Based Computers III*, pages 255–264, 1997.
- [OR99] M. Oghihara and A. Ray. Biomolecular computing—recent theoretical and experimental advances. *SIGACT News*, 30(2):22–30, 1999.
- [ORS97] M. Oghihara, A. Ray, and K. Smith. Biomolecular computing—a shape of computation to come. *SIGACT News*, 28(3):2–11, 1997.
- [Rob65] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the Association for Computing Machinery*, 12(1):23–41, January 1965.
- [ROMJ97] B. B. Rosenbaum, F. Oaks, S. Menchen, and B. Johnson. Improved single-stranded DNA sizing accuracy in capillary electrophoresis. *Nucleic Acids Research*, 25:3925–3929, 1997.
- [Sch99] U. Schöning. A probabilistic algorithm for k -SAT and constraint satisfaction problems. In *Proceedings of 40th Symposium on Foundations of Computer Science*, pages 410–414. IEEE Computer Society Press, Los Alamitos, CA, 1999.
- [SFM89] J. Sambrook, E. F. Fritsch, and T. Maniatis. *Molecular Cloning: a Laboratory Manual*. Cold Spring Harbor Press, NY, 2nd edition, 1989.

Solving Computational Learning Problems of Boolean Formulae on DNA Computers

Yasubumi Sakakibara

Department of Information Sciences, Tokyo Denki University,
Hiki-gun, Saitama 350-0394, Japan
(E-mail: yasu@j.dendai.ac.jp)

Abstract. We apply a DNA-based massively parallel exhaustive search to solving the computational learning problems of DNF (disjunctive normal form) Boolean formulae. Learning DNF formulae from examples is one of the most important open problems in computational learning theory and the problem of learning 3-term DNF formulae is known as intractable if $RP \neq NP$. We propose new methods to encode any k -term DNF formula to a DNA strand, evaluate the encoded DNF formula for a truth-value assignment by using hybridization and PCR, and find a consistent DNF formula with the given examples. By employing these methods, we show that the class of k -term DNF formulae (for any constant k) and the class of general DNF formulae are efficiently learnable on DNA computer.

1 Introduction

The massively parallel search is a very useful method found in DNA computing and has much potential to solve many computational hard (NP-complete) problems. Adleman has employed the massively parallel search for solving the Hamiltonian path problem [1] and Lipton has further developed it to the satisfiability problem [7]. Two fundamental techniques are used in their algorithms:

1. design an assembly graph for generating an exponential number of all possible candidates for the solutions,
2. extract a correct solution through biological operations in efficient time.

In this paper, we apply a DNA-based massively parallel exhaustive search to solving the computational learning problems of DNF (disjunctive normal form) Boolean formulae. Computational learning theory is a new and active research area in computer science and artificial intelligence to examine formal models of inductive inference, discover the common methods underlying efficient learning algorithms and identify the computational hardness to learning. Learning DNF Boolean formulae from examples is a central topic in computational learning theory.

The problem of learning Boolean formulae in computational learning theory is that we first assume the target Boolean formula α_* on $X_n = \{x_1, x_2, \dots, x_n\}$

which is unknown to the learning algorithm, and second a truth-value assignment $a \in \{0, 1\}^n$ and the truth-value of the target Boolean formula α_* for the assignment a , that is, $\alpha_*(a)$, are given to the learning algorithm. We call the pair $(a, \alpha_*(a))$ an *example* and usually an enough number of examples are given to the learning algorithm. Then the problem is to correctly and efficiently identify the target Boolean formula α_* only from those given examples. The problems for learning DNF formulae from examples are computationally hard problems. The problem of learning 3-term DNF formulae is known as intractable if $RP \neq NP$ [9,6], and the problem of learning the class of general DNF formulae remains as one of the most important open problems in computational learning theory.

In order to solve these computationally hard problems of learning DNF Boolean formulae on DNA computers, we propose new methods to encode any k -term DNF formula to a DNA strand, evaluate the encoded DNF formula for truth-value assignments by using hybridization and PCR, and find the consistent DNF formulae with the given examples. Unlike Lipton's solution of the satisfiability problem for Boolean formulae, we encode DNF Boolean formulae to DNA strands, put those DNA strands into the test tube and the procedure consisting of biological operations is executed on the test tube to evaluate DNA strands for truth-value assignments. The key idea is our encoding method to encode any terms (conjunctions of literals on X_n) to DNA single-strands containing *stopper and marker sequences* and concatenate those DNA strands to represent DNF Boolean formulae so that the evaluation of the DNA strands for a truth-value assignment can be efficiently done using the biological operations of hybridization and PCR. By employing these methods, we show that the class of k -term DNF formulae (for any constant k) and the class of general DNF formulae are efficiently learnable on DNA computer.

We also shortly discuss parallel computational aspects and error-resistant computations of our learning algorithms.

2 Computational Learning of Boolean Formulae

Human beings have an ability to learn new concepts without explicit programming. In the machine learning literature, this ability is called *inductive inference* or *learning from examples*. Computational learning theory is a new and active research area in computer science and artificial intelligence to examine formal models of inductive inference, discover the common methods underlying efficient learning algorithms and identify the computational hardness to learning. The work on computational learning research is done by formally modeling the activity of learning and characterizing what can or cannot be learned with respect to some precisely defined notion of learnability. The study of computational learning also has practical motivations such as acquiring human experts' knowledges and synthesizing boolean circuits from input-output pairs.

In this paper, we employ the Valiant's learnability model [11] and consider the Boolean formulae as the target concepts to be learned.

We assume that there are n Boolean variables (or attributes) and we denote the set of such variables as $X_n = \{x_1, x_2, \dots, x_n\}$. A *truth-value assignment* $a = (b_1, b_2, \dots, b_n)$ is a mapping from X_n to the set $\{0, 1\}$ or a binary string of length n where $b_i \in \{0, 1\}$ for $1 \leq i \leq n$. A *Boolean function* is defined to be a mapping from $\{0, 1\}^n$ to $\{0, 1\}$. *Boolean formulae* are useful representations for Boolean functions. The simplest Boolean formula is just a single variable. Each variable x_i ($1 \leq i \leq n$) is associated with two *literals*: x_i itself and its negation $\neg x_i$. A *term* is a conjunction of literals. A Boolean formula is in *disjunctive normal form* (DNF, for short) if it is a disjunction of terms. Every Boolean function can be represented by a DNF Boolean formula. For any constant k , a k -term DNF formula is a DNF Boolean formula with at most k terms. We denote the truth value of a Boolean formula β for an assignment $a \in \{0, 1\}^n$ by $\beta(a)$.

The Valiant's learnability model is the distribution-independent model for learning from random examples and called *probably approximately correct learning model* (PAC model, for short). First, we assume a target Boolean formula α_* on X_n to be learned. An *example* of α_* is a pair (a, l) where a is an assignment in $\{0, 1\}^n$ and l is the truth-value of the target Boolean formula α_* for the assignment a , that is, $l = \alpha_*(a)$. An example (a, l) is called a *positive example* of α_* if $l = 1$ and called a *negative example* if $l = 0$. A *sample* is a finite set of positive and negative examples of α_* . The *size* of a sample S is the number of examples in it. A Boolean formula β is said to *agree with* an example (a, l) if $\beta(a) = l$. A Boolean formula is *consistent* with the given sample if it agrees with all examples in the sample.

We assume that there is an unknown and arbitrary probability distribution D on the domain $\{0, 1\}^n$. The probability of assignment $a \in \{0, 1\}^n$ with respect to D is denoted $\Pr_D(a)$. Random samples are assumed to be drawn independently from the domain $\{0, 1\}^n$ according to this probability distribution D on $\{0, 1\}^n$. A learning algorithm takes a randomly drawn sample as input and conjectures some Boolean formula β on X_n . The success of learning is measured by two parameters, the accuracy parameter ϵ and the confidence parameter δ . We define a notion of the difference between two Boolean formulae α and β with respect to the probability distribution D as

$$d(\alpha, \beta) = \sum_{\alpha(a) \neq \beta(a)} \Pr_D(a).$$

The *error* of a Boolean formula β with respect to the target Boolean formula α_* is $d(\beta, \alpha_*)$. A successful learning algorithm is one that with high probability finds a Boolean formula whose error is small. The notion of *polynomial learnability* in PAC model is formally defined as follows:

A class C_n of Boolean formulae over X_n is *polynomially PAC learnable* if there exists a learning algorithm A for C_n such that for any ϵ and δ , for any target Boolean formula $\alpha_* \in C_n$, and for any distribution D on $\{0, 1\}^n$, when A is given as input a randomly drawn sample S of size

polynomial in n , ϵ and δ , the algorithm A outputs a Boolean formula $\beta \in C_n$ such that $d(\beta, \alpha_*) \leq \epsilon$ with probability at least $1 - \delta$, and runs in time polynomial in the size of S .

When learning the class of general DNF formulae, the sample size and the time of learning algorithm is allowed to be polynomial in n , ϵ , δ , and further the *size* of the target DNF formula α_* where the size of DNF formula β is the number of terms in β .

The following result has been shown for the polynomial learnability of Boolean formulae [2]:

For a class C_n of Boolean formulae, if there exists a polynomial-time learning algorithm that produces a Boolean formula *consistent* with the given sample S , then the class C_n is polynomially PAC learnable.

For the class of k -term DNF formulae, a randomly drawn sample S of size greater than

$$m \geq \frac{kn}{\epsilon} \ln \left(\frac{3}{\delta} \right)$$

is shown to be enough for the consistent learning algorithm, and for the class of general DNF formulae, a randomly drawn sample S of size greater than

$$m \geq \frac{\text{size}(\alpha_*)n}{\epsilon} \ln \left(\frac{3}{\delta} \right)$$

is shown to be enough for the consistent learning algorithm that produces a DNF formula of *smallest* size consistent with S , where $\text{size}(\alpha_*)$ is the size of the target DNF formula α_* . Nevertheless, the problems for learning DNF formulae from given examples are computationally hard problems. The problems of learning the class of 3-term DNF formulae and learning the class of k -term DNF formulae for any constant $k \geq 2$ have been shown to be intractable if $RP \neq NP$ [9], and the problem of learning the class of general DNF formulae remains as one of the most important open problems in computational learning theory.

3 Learning Algorithms on DNA Computers

In this section, we provide the methods to encode k -term DNF formulae to DNA strands, evaluate the encoded DNF formula for a truth-value assignment, and find a consistent DNF formula with the given sample.

3.1 Algorithm to Evaluate k -Term DNF Formulae

To evaluate a k -term DNF formula β on n Boolean variables $X_n = \{x_1, x_2, \dots, x_n\}$, we use the following simple algorithm:

Algorithm A:

Let $a = (b_1, b_2, \dots, b_n)$ be a truth-value assignment on X_n ,
and $\beta = t_1 \vee t_2 \vee \dots \vee t_m$ be a given DNF formula, where t_j ($1 \leq j \leq m$)
is a term on X_n .

Step 1. For each i ($1 \leq i \leq n$),

Step 1.1. If $b_i = 0$ then let $z = x_i$, and if $b_i = 1$ then let $z = \neg x_i$.

Step 1.2. If any term t_j ($1 \leq j \leq m$) contains z , then delete the
term t_j from β .

Step 2. If there remains at least one term in β , then the truth-value of
 β for the assignment a is 1. Otherwise, the truth-value is 0.

The algorithm A eliminates all terms whose truth-values become 0 for the
assignment a , and if any terms t remain in β , the truth-values of the terms
 t become 1 for a and hence the truth-value of β becomes 1 for a . Thus, it
is easily verified that this algorithm correctly calculates the truth-value of the
DNF formula β for the assignment a .

3.2 Implementing the Evaluation Algorithm on DNA Computer

We implement the algorithm A using DNA strands and biological operations.

First, we encode a k -term DNF formula β into a DNA single-strand as follows:

Let $\beta = t_1 \vee t_2 \vee \dots \vee t_k$ be a k -term DNF formula.

(1) For each term t in the DNF formula β , we use the DNA single strand
of the form:

$$5' - \text{stopper} - \text{marker} - \text{lit}_1 - \dots - \text{lit}_n - 3'$$

where lit_i ($1 \leq i \leq n$) is the encoded sequence for x_i if the term t
contains x_i , the sequence for $\neg x_i$ if t contains $\neg x_i$, or the sequence for
empty if t contains neither. The **stopper** is a *stopper sequence* for the
polymerization stop that is a technique developed by Hagiya et al. [4].
The **marker** is a special sequence for an extraction used later at the
evaluation step.

(2) We concatenate all of these sequences encoding terms t_j ($1 \leq j \leq k$)
in β . Let denote the concatenated sequence encoding β by $e(\beta)$.

For example, the 2-term DNF formula $(x_1 \wedge x_2) \vee (\neg x_3)$ on three variables
 $X_3 = \{x_1, x_2, x_3\}$ is encoded as follows and illustrated in Figure 1.

$$5' - \text{marker} - x_1 - x_2 - \text{empty} - \text{stopper} - \text{marker} - \text{empty} - \text{empty} - \neg x_3 - 3'$$

Second, we put the DNA strand $e(\beta)$ encoding the DNF formula β into the
test tube and do the following biological operations to evaluate β for the truth-
value assignment $a = (b_1, b_2, \dots, b_n)$.

MARKER	x_1	x_2	EMPTY	STOPPER	MARKER	EMPTY	EMPTY	$\neg x_3$
--------	-------	-------	-------	---------	--------	-------	-------	------------

Fig. 1. The DNA strand encoding the DNF formula $(x_1 \wedge x_2) \vee (\neg x_3)$.

Algorithm $B(T, a)$:

(1) Let the test tube T contain the DNA single-strand $e(\beta)$ for the DNF formula β .

(2) Let $a = (b_1, b_2, \dots, b_n)$ be the truth-value assignment. For each b_i ($1 \leq i \leq n$), if $b_i = 0$ then put the Watson-Crick complement \bar{x}_i of the sequence encoding x_i into the test tube T , and if $b_i = 1$ then put the complement $\neg x_i$ of $\neg x_i$ into T .

(3) Cool down the test tube T for annealing these complements to complementary substrands in $e(\beta)$.

(4) Apply the Polymerase Chain Reaction (PCR) to the test tube T with these annealed complements as the primers. As a result, if the substrand for a term t_i in β contains a literal lit_i and the bit b_i assigns 0 to lit_i (that is, if b_i is 0 then the truth-value of lit_i equal to x_i becomes 0, and if b_i is 1 then the truth-value of lit_i equal to $\neg x_i$ becomes 0), then the complement $\overline{lit_i}$ of the substrand lit_i has been put at Step (2) and is annealed to lit_i . PCR extends the primer $\overline{lit_i}$ and the subsequence for the marker in the term t_i becomes double-stranded, and the extension stops at the stopper sequence. Otherwise, the subsequence for the marker remains single-stranded. This means that the truth-value of the term t_i is 1 for the assignment a .

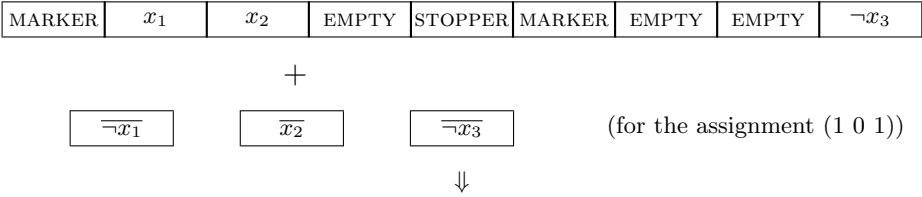
(5) Extract the DNA (partially double-stranded) sequences that contains single-stranded subsequences for markers. These DNA sequences represent the DNF formulae β whose truth-value is 1 for the assignment a .

The figure 2 illustrates the behavior of the algorithm B for $\beta = (x_1 \wedge x_2) \vee (\neg x_3)$ and a truth-value assignment $a = (101)$ on $X_3 = \{x_1, x_2, x_3\}$, and the figure 3 illustrates for β and a truth-value assignment $a = (111)$. The truth-value of β is 0 for the assignment $a = (101)$ and 1 for the assignment $a = (111)$.

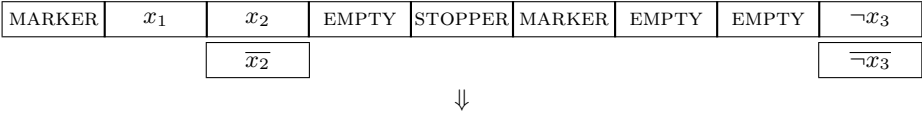
3.3 The Consistent Learning Algorithm on DNA Computer

Now we present the efficient DNA-based algorithm for learning the class of k -term DNF formulae for any constant k . We employ an exhaustive search for finding the consistent hypotheses of DNF formulae with the given sample.

The DNA-based consistent learning algorithm for k -term DNF formulae consists of two steps:



Annealing:

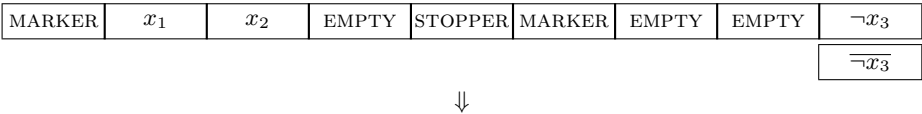


PCR (extension):



Fig. 2. (upper:) For the assignment (101), the Watson-Crick complements $\overline{\neg x_1}$, $\overline{x_2}$ and $\overline{\neg x_3}$ of the encodings for $\neg x_1$, x_2 and $\neg x_3$ respectively are put to the test tube and (middle:) $\overline{x_2}$ and $\overline{\neg x_3}$ are annealed to the DNA strand encoding the formula $(x_1 \wedge x_2) \vee (\neg x_3)$. (lower:) PCR extends the primers $\overline{x_2}$ and $\overline{\neg x_3}$ and both markers become double-stranded.

Annealing:



PCR (extension):



Fig. 3. (upper:) For the assignment (111), the Watson-Crick complements $\overline{\neg x_1}$, $\overline{\neg x_2}$ and $\overline{\neg x_3}$ of the encodings for $\neg x_1$, $\neg x_2$ and $\neg x_3$ respectively are put to the test tube and $\overline{\neg x_3}$ is annealed to the DNA strand for $(x_1 \wedge x_2) \vee (\neg x_3)$. (lower:) PCR extends the primer $\overline{\neg x_3}$, and the right marker becomes double-stranded and the left marker remains single-stranded.

Algorithm $C(k, S)$:

Step 1. Produce all possible k -term DNF formulae and put them to the test tube T_h . This procedure can be done using a similar technique to Adleman's experiment [1] and Lipton's one [7] as follows:

Consider the directed assembly graph in Figure 4. Any path from v_{in} to v_{out} corresponds to a DNA strand encoding a k -term DNF formula. For the directed graph for k -term DNF formulae, there are 3^{kn} paths from v_{in} to v_{out} which represents all possible k -term DNF formulae. Then by careful encoding the vertices and edges with DNA single-strands for the directed graph and doing the same biological operations as Adleman's, the soup in the resulting test tube contains DNA strands which encodes arbitrary paths through the graph.

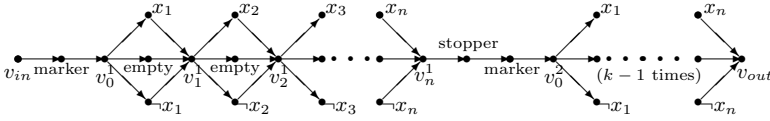


Fig. 4. An assembly graph for generating k -term DNF formulae.

Remark: The assembly graph in Figure 4 may produce a term consisting of only *empty* sequences:

$$5' - \text{stopper} - \text{marker} - \underbrace{\text{empty} - \dots - \text{empty}}_{n \text{ times}} - 3'$$

Hence before going to Step 2, we eliminate all DNF formulae which contain such terms. This can be done by the following procedure: We put the complementary sequences for all literals $x_1, x_2, \dots, x_n, \neg x_1, \neg x_2, \dots, \neg x_n$ on X_n into the test tube T , anneal them and apply PCR to T . If the subsequence for the marker in a term remains single-stranded, it means that the term consists of only empty sequences. We extract and eliminate them.

Step 2. Let T_h be the test tube containing all possible k -term DNF formulae. Let S be the given sample.

- Set $T = T_h$.
- For each positive example $(a, 1)$ in S , run the algorithm $B(T, a)$ and extract DNF formulae whose truth-value is 1.
- For each negative example $(a, 0)$ in S , run the algorithm $B(T, a)$ and eliminate DNF formulae whose truth-value is 1 and extract DNF formulae whose truth-value is 0.

Step 3. Any DNA strands remaining in the final test tube are consistent DNF formulae with the given sample. Detect one of them and output it.

It is clear that the algorithm $C(k, S)$ runs in time (biosteps) linear in the size of the given sample S . Hence it implies that the class of k -term DNF formulae for any constant k is polynomially PAC learnable on DNA computers.

By the duality, it is also easily confirmed that the class of k -clause CNF (conjunctive normal form) formulae for any constant k is polynomially PAC learnable on DNA computers, where a *clause* is a disjunction of literals on X_n .

3.4 DNF Formulae Are Efficiently Learnable on DNA Computer

By employing the algorithm $C(k, S)$, we show that the class of general DNF formulae are efficiently learnable. Given a randomly drawn sample of size greater than $\frac{\text{size}(\alpha_*)n}{\epsilon} \ln \left(\frac{3}{\delta} \right)$ for the target DNF formula α_* , we iteratively use the algorithm $C(k, S)$ for each k from $k = 1$ to possibly $k = \text{size}(\alpha_*)$ until finding a consistent DNF formula with S .

Algorithm $D(S)$:

Let S be the given sample.

Set $k = 1$;

while (no DNF formula consistent with S is found) **do** {
 call the algorithm $C(k, S)$;
 $k = k + 1$;
};

Output a DNF formula found consistent with S .

It is clear that the algorithm $D(S)$ terminates at least by $k = \text{size}(\alpha_*)$ and produces a DNF formula of *smallest* size consistent with S , and runs in time (biosteps) proportional to the size of the given sample S . Hence it implies that the class of general DNF formulae is polynomially PAC learnable on DNA computers.

4 Parallel and Probabilistic Computational Aspects

It is addressed that there are two types of parallelism in the DNA computation process of constructing superstructures from substructures. That is, the global parallelism is that more than one construction of superstructure can proceed in parallel in the test tube, and the local parallelism is that growth on each individual superstructure may occur at many locations simultaneously.

In our algorithms for evaluating the encoded sequence $e(\beta)$ for a k -term DNF formula $\beta = t_1 \vee t_2 \vee \cdots \vee t_k$, the evaluation of every term t_i ($1 \leq i \leq k$) in β for a truth-value assignment will proceed independently and simultaneously, that is, the annealings and PCRs will happen in parallel at each term t_i (*local parallelism*). On the other hand, our learning algorithm makes use of the massively parallel search (*global parallelism*).

Here, we also study *error-resistant* DNA computations for our learning algorithm. In the probabilistic model of learning (PAC model) that we are working on, there are several researches for probabilistic noise-tolerant learning methods [35,10]. By employing such noise-tolerant methods, we construct error-resistant DNA computations for learning Boolean formulae. Here we consider the *approximate* consistent learning algorithm studied in [35].

The *approximate consistent* learning algorithm is a polynomial-time learning algorithm that produces a Boolean formula consistent with *at least a fraction* $1 - \epsilon/2$ of the given sample S *with probability at least* $1 - 2\delta/3$. (Note that the constants “ $1/2$ ” and “ $2/3$ ” can be changed to any real values between 0 and 1.) For the class of k -term DNF formulae, a randomly drawn sample S of size greater than

$$m \geq \frac{8kn}{\epsilon} \ln \left(\frac{9}{\delta} \right)$$

is shown to be enough for the approximate consistent learning algorithm. We apply this result to an error-resistant DNA computation for learning k -term DNF formulae.

We consider the following three types of errors in the learning algorithm $C(k, S)$:

1. At Step 1, it fails to generate some of k -term DNF formulae,
2. At Step 2, it fails to extract DNF formulae whose truth-value is 1 for a positive example $(a, 1)$,
3. At Step 2, it fails to eliminate DNF formulae whose truth-value is 1 for a negative example $(a, 0)$.

The errors of type 1 and 2 causes the algorithm $C(k, S)$ to fail to find a consistent DNF formula. The error of type 3 causes the algorithm $C(k, S)$ to fail to eliminate a DNF formulae which does not agree with the example $(a, 0)$. By applying the approximate consistent learning algorithm, the total amount of errors of type 1 is allowed at most $\delta/3$, the total amount of errors of type 2 is also allowed at most $\delta/3$, and the total amount of errors of type 3 is allowed at most $\epsilon/2$. The details will be reported.

5 Conclusions

We have proposed new methods to encode DNF formulae to DNA single-strands and evaluate the encoded DNF formulae in the test tube for a truth-value assignment by using hybridization and PCR. By employing these methods, we have presented the DNA-based learning algorithms for polynomially learning the class of k -term DNF formulae (for any constant k) and the class of general DNF formulae that fully make use of massively parallel exhaustive search in the test tube.

Most related is the work done by Hagiya et al. [4] and their method called “whiplash PCR” [12]. Two important techniques in whiplash PCR have been developed: (1) PCR activity can be conveniently terminated by a *stopper sequence*

in the template (which is a technique used also in this paper), and (2) if the 3' end of a DNA strand serves as the same strand's primer, then an individual DNA molecule can be a self-contained computational unit. They have described how whiplash PCR can be used to solve the problem of learning μ -formulae from the given sample. While whiplash PCR is a very powerful technique (Winfree [12] has presented many $O(1)$ computing solutions for NP-complete problems using whiplash PCR), our proposed methods are much simpler and can be applied to the general DNF Boolean formulae and would be more adequate for computational learning problems.

Our future work is to verify the feasibility of our methods by in-vitro experiments.

Since we take a naive massively parallel exhaustive search, we need 3^{kn} DNA strands generated in test tube for learning k -term DNF formulae. This amount of DNA strands contain much redundancy. Hence, other future work is to improve the space efficiency.

Acknowledgments. We would like to thank Satoshi Kobayashi, Masami Hagiya, Kensaku Sakamoto and Takashi Yokomori for their useful comments. This work is supported in part by "Research for the Future" Program No. JSPS-RFTF 96I00101 from the Japan Society for the Promotion of Science.

References

1. L. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266, 1994, 1021 – 1024.
2. A. Blumer, A. Ehrenfeucht, D. Haussler, M. K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM*, 36, 1989, 929 – 965.
3. R. Board, L. Pitt. On the necessity of Occam algorithms. *Theoretical Computer Science*, 100, 1992, 157 – 184.
4. M. Hagiya, M. Arita, D. Kiga, K. Sakamoto, S. Yokoyama. Towards parallel evaluation and learning of Boolean μ -formulas with molecules. In *Proc. of Third Annual Meeting on DNA Based Computers*, 1997, 105 – 114.
5. M. Kearns, M. Li. Learning in the presence of malicious errors. In *Proc. of 20th Annual ACM Symposium on Theory of Computing*, ACM, 1988, 267–279.
6. M. Kearns, U. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, Massachusetts, 1994.
7. R. J. Lipton. DNA solution of hard computational problems. *Science*, 268, 1995, 542 – 545.
8. Gh. Păun, G. Rozenberg, A. Salomaa. *DNA Computing*. Springer-Verlag, Heidelberg, 1998.
9. L. Pitt, L. G. Valiant. Computational limitations on learning from examples. *Journal of the ACM*, 35, 1988, 965 – 984.
10. Y. Sakakibara. Noise-tolerant Occam algorithms and their applications to learning decision trees. *Machine Learning*, 11, 1993, 37 – 62.
11. L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27, 1984, 1134 – 1142.
12. E. Winfree. Whiplash PCR for $O(1)$ computing. In *Proc. of Fourth Annual Meeting on DNA Based Computers*, 1998.

The Fidelity of Annealing-Ligation: A Theoretical Analysis

John A. Rose¹ and Russell J. Deaton²

¹ Institute of Physics, The University of Tokyo, Komaba 3-8-1, Meguro-ku, Tokyo, 153, Japan, johnrose@genta.c.u-tokyo.ac.jp

² Department of Computer Science and Computer Engineering, The University of Arkansas, Fayetteville, Ark, 72701, USA, rdeaton@uark.edu

Abstract. Understanding the nature of the error propagation through successive biosteps is critical to modeling the overall fidelity of computational DNA architectures. In this work, the fidelity of the compound biostep annealing-ligation is discussed in the limit of zero dissociation, within the framework of a simple statistical thermodynamic model. For simplicity, a DNA ligase of ideal infidelity is assumed, with its error behavior taken as bounding that of real DNA ligases. The derived expression for the fidelity of annealing-ligation indicates that the error coupling is both strong and dependent on sequence. Estimates of the fidelities of annealing and annealing-ligation have also been calculated for various encodings of Adleman's graph, assuming a staggered zipper model of duplex formation. Results indicate the necessity of including information regarding the specific free energies and/or occupancies of accessible duplex states, in addition to information based purely on sequence comparison.

1 Introduction

The experimental feasibility of biomolecular computing was demonstrated in 1994 by Leonard Adleman [1]. In his classic experiment, the solution to a simple instance of the Hamiltonian Path Problem (HPP) was successfully computed *in vitro*, employing only a mixture of ssDNA molecules and a series of protocols adapted from molecular biology. Although the problem instance solved was computationally modest, the number of hybridization and ligation reactions that occurred during the course of the computation ($\approx 10^{14}$) suggested that the solution of much larger instances should, in principle, be feasible. As noted in the original proposal, the potential of the algorithm to effect roughly 10^{20} separate ligation events using physically reasonable DNA volumes compares quite favorably with the number of basic operations (roughly 10^{12} operations/second) achievable by the fastest supercomputers.

The initial excitement generated by the massive parallelism inherent in DNA computation has been moderated somewhat by the difficulty experienced by researchers attempting to repeat and scale Adleman's methodology [2]. As a result, a variety of studies have focused upon the propensity of the various biotechniques

to deviate from the ideal behaviors assumed by the computational model [3-7]. In lieu of a complete error analysis, a popular approach has been to attempt to remedy DNA computation by means of iteration, modification, or complete elimination of a suspected error-prone biotechnology. As a result, the computational power of an impressive array of alternative DNA computational architectures has now been theoretically explored.

The scaling and standardization of biomolecular computing techniques requires that the fidelities of the various biosteps be quantified. Such a characterization not only aids in eliminating uncertainty, but also provides a basis for engineering reliable and efficient computations. The fidelity and efficiency of many of the biotechniques which are critical to DNA computation, such as solid phase synthesis [8], DNA ligation [6,7], and DNA polymerization [9], have already been subjected to individual scrutiny. However, because the result of each phase in a biomolecular computation depends upon the results of the previous step, the tendency for post-annealing enzyme-based biosteps (*i.e.*, ligation, polymerization, digestion) to propagate or generate error is critically dependent not only upon the overall fidelity of the annealing reactions, and that of the enzyme employed, but also upon the identity and relative abundance of the actual set of dsDNA molecules formed during annealing. As a concrete example, in the annealing biostep of Adleman's original architecture, the formation of sets of adjacent DNA duplexes is accomplished by parallel hybridization. A subsequent ligation reaction then acts upon the set of dsDNA precursors formed during the annealing biostep. An analysis of the error potential of the compound process formed by annealing followed by a post-annealing ligation, therefore, must begin with a quantitative consideration of the particular hybridization reactions which dominate the initial annealing phase. Rather than being independent, and hence separable, the fidelity of successive biosteps is intrinsically coupled.

In previous work, the statistical thermodynamic theory of the DNA helix-coil transition was used to derive a limiting expression for the potential for hybridization error for a mixture of DNA oligonucleotides [10]. Following brief, motivating reviews of the computational incoherence and the fidelity of various DNA ligases, the theory of statistical thermodynamics is used to discuss the error behavior of successive annealing and ligation biosteps, in the limit of zero dissociation. The coupling of error between the annealing and ligation biosteps is shown to be strong in nature, in the sense that the overall fidelity of coupled annealing-ligation is not expressible as a product of the fidelities of the individual biosteps, but rather, includes a

factor which depends upon both the specific nature of the DNA ligase and the specific sequences of molecules designed for the problem (*i.e.*, the encoding).

2 The Computational Incoherence, ξ

The theory of DNA statistical thermodynamics has recently been used to discuss the hybridization error probability for an annealing mixture of single stranded (ss) DNAs, and to derive a simple expression for the error behavior of the mix-

ture in the limit of zero (or more generally, uniform) dissociation [10]. In order to ensure a high occupancy of potential hairpin-forming species in an alternative dimerized form, attention was also restricted to the limit of high counterion concentration. In addition, each ssDNA species was assumed to have at least one high-affinity partner, in order to allow for the uniform attainment of low dissociation, in the limit of low reaction temperature T_{rx} . The resulting expression, dubbed the computational incoherence, is given by:

$$\xi \approx \frac{\sum_{i,j \geq i} C_i^\circ C_j^\circ \sum_k \delta_{ijk} Z^{ijk}}{\sum_{i,j \geq i} C_i^\circ C_j^\circ \sum_k Z^{ijk}}, \quad (1)$$

where Z^{ijk} is the statistical weight of hybridized configuration k between DNA species i and j , and δ_{ijk} is a function which equals 1/0 if the hybridized configuration $\{i,j,k\}$ is forbidden/allowed by the hybridization rules of the computation. The sums are taken over all pairs of ssDNA species, i and j , and over all double stranded configurations k which are accessible to the indicated hybridized species, i and j . For the case in which all initial concentrations of ssDNA reactants are approximately equal, ξ reduces to the simpler form:

$$\xi \approx \frac{\sum_{i,j \geq i} \sum_k \delta_{ijk} Z^{ijk}}{\sum_{i,j \geq i} \sum_k Z^{ijk}} = \frac{\sum_{i,j \geq i} Z_e^{ij}}{\sum_{i,j \geq i} Z_c^{ij}}, \quad (2)$$

where the sum over the statistical weights of all double-stranded configurations between species i and j has been recognized as the conformal partition function, Z_c^{ij} for i and j [11], and Z_e^{ij} is identified as the sum over the statistical weights, or equivalently the equilibrium constants, of all erroneous configurations between i and j . An immediate consequence of this expression is the prediction that, at least in the absence of extreme variations in the equilibrium concentrations of ssDNA species, high fidelity is achieved by jointly minimizing the equilibrium constants of erroneous hybridized species.

The best method of estimating the statistical weights of the set of accessible configurations is a separate, and critical issue. In practice, evaluation depends upon the model of duplex formation employed. The use of a general model, in which every double-stranded configuration is regarded to have significant occupancy, is most accurate. The exponential number of configurations, in terms of length, which must be independently assessed, however poses significant practical difficulties. In [10], attention was restricted to mixtures composed of short ssDNAs, facilitating the use of a statistical, or “staggered” zipper model. In the classic staggered zipper model, configurations which require multiple nucleation events are regarded to have negligible occupancy [11]. Each relevant configuration is then characterized by a single, Watson-Crick duplex. The statistical weight of each dsDNA configuration is related to the standard Gibbs free energy of formation of the duplex from isolated single strands, ΔG_{ijk}° , by the expression,

$$Z^{ijk} = \exp\left(-\frac{\Delta G_{ijk}^\circ}{RT_{rx}}\right), \quad (3)$$

where R is the molar gas constant and T_{rx} is the Kelvin temperature [11]. According to the nearest-neighbor model of duplex formation [12], ΔG° is estimated by:

$$\Delta G^\circ = \sum_{nn} \Delta G^\circ(nn) + \sum_{ends} \Delta G^\circ(\text{init}) + \Delta G^\circ(\text{sym}), \quad (4)$$

where first term accounts for the free energy of base-pair stacking, and is a sum over the set of nearest-neighbor doublets, nn , which are contained in the configuration's lone duplex, the second term is a length-independent initiation parameter which estimates both the free energy of strand association ($\Delta G_{str.as.}^\circ$) and the effects of unwinding at each duplex terminus, and the third term is an entropic penalty applied only to palindromic duplexes. A set of nearest-neighbor parameters appropriate for the prediction of the free energy of both oligonucleotides and polynucleotides was reported in [12]. The use of a staggered zipper model, combined with a nearest-neighbor model of duplex formation, allows a polynomial-time estimation of ξ vs. T_{rx} .

3 The Fidelity of DNA Ligases

The joining of the sugar-phosphate backbones of a pair of ssDNAs, resulting in the formation of a single backbone, is known as DNA ligation. Because of the importance of ligation in DNA repair and recombination [13], the ability to produce at least one form of DNA ligase is common to all organisms, whether prokaryotic, eukaryotic, or viral. Conceptually, joining is accomplished by the formation of a phosphodiester bond between directly adjacent 3'-hydroxyl and 5'-phosphoryl groups. Although dsDNA molecules with either cohesive or blunt ends may serve as a substrate for ligation, cohesive end ligation is much more efficient [14]. Ligation is therefore usually *template-mediated*, in that the pair of ssDNAs to be joined must be held in proximity by hybridization to a third splinting ssDNA molecule. In any case, substrate DNAs must be duplex. Single-stranded DNA does not serve as substrate for DNA ligase [15].

The fidelity of various DNA ligases has been examined *in vitro*. One of the most popular DNA ligases, and the DNA ligase used by Adleman [1] is the DNA ligase induced in *Escherichia coli* following infection with the T4 bacteriophage, or the T4 DNA ligase. A well known drawback to using T4 DNA ligase is its ability to efficiently catalyze blunt-end ligation [14]. This ability is so pronounced that an unpaired nucleotide protruding at the 3' or 5' end of a complementary strand does not prevent efficient blunt-end ligation [16]. In addition, T4 DNA ligase has been demonstrated to efficiently catalyze ligation reactions containing a wide variety of mismatched and "improperly" hybridized substrate oligonucleotides. Experiments indicate that T4 DNA ligase seals nicks with 3' or 5' apurinic sites [16], a 1 nucleotide gap [16], 3' or 5' terminal A-A or T-T mismatches [16], 5' terminal G-T mismatches [17], 3' terminal C-A, C-T, T-G, T-T, T-C, A-C, G-G, or G-T mismatches [18], and a variety of internal mismatches [19]. On the other hand, T4 ligase does display some ability to reject mismatched substrate molecules, and is roughly 5 times more efficient at sealing

a mismatch at the 3' terminus [20]. This superior ability to discriminate versus a 5' (relative to 3') terminal mismatch is atypical among DNA ligases. In addition, recent studies suggest successful discrimination by T4 DNA ligase versus substrate molecules containing multiple terminal mismatches [7]. It is noteworthy that the efficiency of T4 DNA mismatch ligase decreases with increasing NaCl concentration [20].

Another viral DNA ligase which has been subjected to scrutiny is that encoded by the Vaccinia virus [21]. Although Vaccinia ligase was observed to display strong discrimination ability versus 1 and 2 nucleotide gaps and 3' purine-purine mismatches, it tolerated a wide variety of other mismatched substrate molecules. Vaccinia ligase was observed to seal substrate molecules containing 3' C-A, C-T, G-T, T-T, and G-T mismatches, 5' C-T, G-T, T-T, A-C, T-C, C-C, G-G, T-G, or A-G terminal mismatches readily [21]. A tendency to discriminate 3' over 5' terminal mismatches is regarded as typical for DNA ligases.

The DNA ligase purified from uninfected *Escherichia coli* has been examined as a possible candidate for use in DNA computation, because of its reported inefficiency at catalyzing blunt-end ligation [6]. This study was particularly interesting because it included an assessment of the tolerance of DNA ligase to a combinatorial mixture containing all possible mismatched molecules of length 12. Although *E. coli* ligase was observed to demonstrate some discrimination ability versus terminal mismatches, it was also observed to tolerate a wide spectrum of internally mismatched and frameshifted substrate molecules. In particular, each successfully ligated molecule was observed to contain an average of 3.3 mismatches. In fact, if frameshifted (bulged) nucleotides are included, successful ligation was observed for substrate molecules containing mismatches at as many as 7 of 12 possible positions.

Several thermostable DNA ligases have been observed to demonstrate an apparent reliability superior to *E. coli* or T4 DNA ligase. The DNA ligase derived from *Thermus aquaticus* (*Taq*) is a case in point. At thermophilic temperatures, *Taq* DNA ligase was observed to display enhanced ability (roughly, from 10 to 100-fold, in terms of final product) to discriminate versus all mismatches studied (terminal 3', only) [22]. In addition, the fidelity displayed was observed to be moderately invariant to salt concentration, in contrast to that of mesophilic ligases. Some caution in interpreting the results, however, is in order. The single experiment reported using *Taq* ligase was performed at 65°C, for a set of oligonucleotides with T_m values in the range from 66 – 70°C. Differences in the total amounts of ligated product for perfectly matched vs. partially mismatched oligonucleotide substrates will therefore reflect the enhanced dissociation of mismatched substrate as well as the discrimination ability characteristic of the enzyme. In addition, only 3'-terminal mismatches were studied. As a result, the discrimination ability of *Taq* DNA ligase to 5'-terminal and internal mismatches remains an open question.

Thermostable DNA ligase derived from *Thermus thermophilus* (*Tth*) has also been reported to demonstrate enhanced reliability (relative to T4 and *E. coli* DNA ligases) at a variety of temperatures. In [23] (65°C), *Tth* ligase was observed

to exhibit a strong ability to discriminate versus substrate molecules containing 3' terminal mismatches, with only GT and TG detected. The observed fidelity versus 5' terminal mismatches, however, was substantially lower. In [24] (37°C), *Tth* ligase was reported to efficiently ligate substrate containing single 5' terminal mismatches, and in addition, to ligate substrate containing tandem 5' terminal mismatches, albeit inefficiently. In addition, *Tth* ligase was observed to ligate substrate molecules having single internal mismatches at any position, although in each case, with reduced efficiency. The most recent study [25], performed at a variety of temperatures [25] (44°, 46°, 48°, and 50°C), is distinguished in that it assessed the ligation fidelity of *Tth* ligase versus a combinatorial mixture of all substrates of length 9. The primary new result was that although *Tth* ligase could successfully ligate substrate molecules containing single and multiple mismatches at any position, a strong bias exists for the occurrence of mismatches containing guanine, and for mismatches occurring at positions 5 and 9, relative to the 3' terminus, which is suggestive of enzyme-specific anchorage requirements.

The fidelities of eukaryotic DNA ligases are less well characterized. For a particular bacterial species, all ligation reactions appear to be catalyzed by the bacterium's single general-purpose DNA ligase. In contrast, the various ligation reactions observed in a eukaryote are often the result of a set of distinct, highly specialized DNA ligases [26]. For instance, at least 5 distinct species of DNA ligase (I-V) have been purified from mammalian cells [27]. As a result of this specialization, a particular eukaryotic DNA ligase is often unable to carry out ligations between arbitrary oligonucleotides [27]. An exception is the yeast, *Saccharomyces cerevisiae*, which has been observed to harbor only a single, general-purpose DNA ligase. Although *S. cerevisiae* DNA ligase has been observed to discriminate against 3' A-G or T-G mismatches as well as a 1 nt gap, its marked efficiency in sealing other mismatches has prompted speculation that the DNA ligase may play a major role in mutation [28].

The literature on the ability of DNA ligases to efficiently perform various types of mismatch ligations is largely qualitative, and is fragmentary in nature. The clearest result is that DNA ligases differ substantially in fidelity, often in a complicated manner. An overall picture, however, has begun to emerge. As noted in [24], the observed fidelity of a given DNA ligase is expected to be due to three factors: (1) The relative duplex instability caused by mismatches; (2) an observed necessity for the particular ligase to make contacts with the DNA duplex at some subset of nucleotides beginning with, and extending away from, the site of closure. Various ligases will therefore be expected to fail to ligate oligos below some threshold length; (3) The need for a correct "duplex-like" geometry of the DNA substrate for recognition by the ligase active site.

The relative contribution of each factor to overall ligase fidelity is an open question. Discrimination anticipated from duplex formation alone, however, is substantially lower than the discrimination by ligases actually observed [25]. This conclusion is supported by the relative ligation rates of mismatches at the 3' and 5' termini. In [23], it was noted that if ligation fidelity were mainly dependent upon the stacking-induced stability of the duplex in the vicinity of the junction,

then duplex stability would be expected to be higher for the 5' terminus than the 3' terminus, given the observation that *Tth* ligase discriminates against 3' mismatches more efficiently. As the theoretical stacking stability was calculated to be lower on the 5' side, however, the observed differential fidelity of *Tth* ligase for mismatches at the 3' terminus vs. those on the 5' terminus is not due to duplex energetics, but rather to a more stringent 3' terminus recognition requirement of the enzyme. A greater fidelity to 3' vs. 5' mismatches has also been observed for virally-derived (Vaccinia virus [21]), prokaryotic (*Taq* [22], *E. coli* [6]), and eukaryotic (*S. cerevisiae* [28]) DNA ligases. As noted earlier, the opposite trend in discrimination (5' over 3') which is exhibited by T4 DNA ligase, however, indicates that a relative 3' over 5' mismatch fidelity is not a general feature of DNA ligases. Another observation which suggests that local base pair stability is not the primary factor influencing ligase fidelity at the site of ligation is that both bacterial and mammalian DNA ligases have a very high fidelity vs. all purine-purine mismatches, including 3' G-A or A-G mismatches, which are among the most stable of the mismatches [18,21,23].

Several studies indicate that the fidelity of DNA ligases to mismatches at positions other than the 5' and 3' termini is neither random nor in accordance with that expected from helix stability considerations [6,24,25]. *Tth* DNA ligase shows mismatch "hot-spots" for mismatches at the middle position and for mismatches at the 5' terminus, while the best fidelity is observed vs. mismatches at nucleotide position 4 (3 nucleotides from the 3' terminus) [25]. Ligation of mismatches at the rest of positions have roughly equal reliability. This set of results is counter to that expected a priori from stability considerations, but is consistent with the observed requirement of DNA ligases for contact points beyond the 3' and 5' termini [25].

4 Bounding the Fidelity of Annealing-Ligation

The computational incoherence (ξ) provides a well-defined statistical measure of the hybridization error potential of a DNA mixture, under the conditions of low dissociation and high counterion concentration. From the perspective of a molecular observer sampling the error state of hybridizations in the mixture, all computationally forbidden modes of hybridization are equivalent, and are therefore grouped into a single error "state" for the calculation of ξ . As noted previously, however, an annealed ensemble is usually intended to serve as precursor to a secondary, enzymatically-implemented biostep (i.e., ligation, in Adleman's algorithm). As a result, the impact of each error mode on the overall reliability of the computation will not be uniform, but rather will acquire an enzyme-dependent context sensitivity. The computational incoherence is therefore not adequate to provide a complete assessment of the reliability of the computational process. In order to relate ξ to the overall fidelity, it is necessary to extend the results of a statistical mechanical analysis of hybridization error to include the impact of a secondary enzymatic process. Although there are numerous instances of such compound processes (i.e., hybridization followed by exonuclease digestion,

DNA ligation, or DNA polymerization), for clarity, attention is restricted to a consideration of ligation error in an annealed DNA mixture at equilibrium.

From the previous discussion on ligase fidelity, it is clear that the potential for error ligation is a function of a variety of variables, including the potential for contiguous error hybridizations in the annealing ensemble, and both the processivity and fidelity of the specific ligase enzyme chosen to catalyze the reaction. It is also clear, however, that as a catalyst, ligase may only catalyze reactions for which there exist a set of precursor molecules. Ligation error potential may therefore be effectively assessed by identifying the subset of error hybridizations most likely to generate ligation error, and estimating the relative abundance of this subset. Likewise, the overall fidelity of the set of ligation reactions occurring in the mixture may then be enforced by encoding to minimize the abundance of this fraction. In the absence of a significant population of error precursor molecules, the fidelity of the various DNA ligases becomes a secondary issue.

In Adleman's architecture, a single ligation event requires two independent hybridization events. However, the analysis of the propensity of an annealed mixture to ligation error may be simplified by recognizing that those hybridization-error configurations which are suitable for ligation with an adjacent non-error (expected) hybridization will experience a greatly enhanced probability of ligation, relative to other error configurations. Let such error ligations, which require only a single hybridization error (in addition to a planned hybridization event) for expression, be termed *first order* ligation errors. In order to facilitate a first-

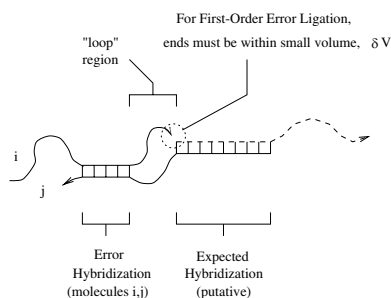


Fig. 1. A First Order Ligation Event

order ligation error, a configuration must satisfy a minimum set of requirements (see Fig. II). First of all, the configuration of one hybrid must contain an error duplex region which does not interfere with participation in one or more additional, expected hybridizations. Note that a particular error mode of hybridization between ssDNA species *i* and *j* may facilitate from zero to two separate first order error ligation events, depending upon the location of the error duplex relative to the midpoints of ssDNA species *i* and *j*. In addition, for successful ligation, appropriate ss regions of the configuration must be oriented in a manner which facilitates a first-order error ligation event. This is simply the requirement that

the relevant free end of the error duplex be in a position allowing formation of a secondary nucleation near the appropriate end of a planned hybridization. As illustrated in Fig. 1, this requires that the two ends for ligation approach to within a small reaction volume, δV . Given the negligible enthalpy of nucleation in aqueous solution, the condition of proximity is considered here to be the primary determinant of ligatability, and the specific nature of the secondary nucleation event (*i.e.*, the specific hydrogen bond formed) is neglected. The potential for formation of stacked base pairs in the “loop” region between the secondary nucleation site and the error duplex can be expected to impact the potential for facilitating a first order ligation error. If the degree of stacking in this region is sufficient to generate an intact duplex, it must be considered as a distinct configuration, and treated separately. The formation of isolated stacks or short stacked segments (each of length ≤ 3 bps) however, is extremely unfavorable due to the size of the cooperativity parameter [29], and can be assumed to make a negligible contribution to the overall statistical weight.

The absence of significant stacking interactions at specific areas in the loop region can also be expected to have an inhibitory impact on the ability of the various DNA ligases to successfully anchor themselves to and ligate substrate molecules, the presence of favorable minimum conditions notwithstanding. Unfortunately, the obvious promiscuity of the various DNA ligases, combined with the fragmentary, qualitative nature of current information on ligase fidelity, obviates any attempt to quantitatively weight candidate configurations for ligatability based on ligase substrate specificity, effectively preventing a meaningful statistical estimate of first order ligation error propensity. On the other hand, by considering the action of a theoretical ultrapotent DNA ligase, which has zero substrate specificity within this loop region, an upper bound on the propensity of each configuration, and thus of a particular encoding, to first order ligation errors may be computed successfully.

The assignment of a total statistical weight, Z_{fle}^{ijk} to the configuration in Fig. 1 relative to that of the unstacked, separated single strands requires some care. The statistical weight of the (error) duplex region, Z^{ijk} may be estimated from the specific duplex composition, using the nearest neighbor model (equation 4) and the parameter set described in [12]. For a configuration containing multiple duplex regions punctuated by an internal loop, an estimate of the overall statistical weight of the internal loop is given by $\sigma^{\frac{1}{2}} f(m) \sigma^{\frac{1}{2}} = \sigma f(m)$, where each factor of $\sigma^{\frac{1}{2}}$ accounts for the loss of a stabilizing stacking interaction at a closing base pair of the loop, in addition to a loop normalization factor. The normalized probability that a loop of m broken base pairs will occupy a configuration appropriate for cyclization, $f(m)$, is estimated by:

$$f(m) = \frac{1.0}{(1 - 1.38^{-0.1m})(m + 1)^{1.7}}, m \geq 4, \quad (5)$$

with cited specific values of $f(m)$ used for $m < 3$ [29]. This suggests the form, $Z_{fle}^{ijk} = \sigma^{\frac{1}{2}} f(m) Z^{ijk}$. However, because a factor of σ is included implicitly in the chain association parameter, $\beta = \exp[-\Delta G_{str.as.}^{\circ}/RT_{rx}]$ [29], the explicit

inclusion $\sigma^{\frac{1}{2}}$ in Z_{fle}^{ijk} is redundant when the initiation parameters of the nearest-neighbor model have been applied to estimate the statistical weight of the duplex and bimolecular association. The correct form of the statistical weight is therefore given by:

$$Z_{fle}^{ijk} = f(m)Z^{ijk}, \quad (6)$$

with $f(m) = 0$ for configurations with zero probability of facilitating the required secondary nucleation.

Let χ be the equilibrium average *number* of first-order error ligation events that a randomly observed hybridization is suitable to facilitate (with an adjacent planned hybridization) in the presence of the hypothetical ultrapotent ligase. An expression for χ in the limit of zero dissociation is obtained by a statistical thermodynamic development parallel to that previously presented in detail for ξ . In the first-order error ligation model, any error hybrid may participate in up to two distinct ligation events. The desired expression is therefore given by:

$$\chi \approx \frac{\sum_{i,j \geq i} \sum_k \delta_{ijk} Z^{ijk} \sum_l f(m_l)}{\sum_{i,j \geq i} Z_c^{ij}} = \frac{\sum_{i,j \geq i} Z_{fle}^{ij}}{\sum_{i,j \geq i} Z_c^{ij}}, \quad (7)$$

where the index l sums over all first order ligatable configurations facilitated by error hybrid ijk and Z_{fle}^{ij} is total statistical weight of all configurations between species i and j which may facilitate a first order error ligation. χ is related to ξ by the expression:

$$\chi \approx \frac{\sum_{i,j \geq i} Z_{fle}^{ij} \sum_{i,j \geq i} Z_e^{ij}}{\sum_{i,j \geq i} Z_e^{ij} \sum_{i,j \geq i} Z_c^{ij}} = \langle N_e^{fle} \rangle, \quad (8)$$

where the bounding quantity, $\langle N_e^{fle} \rangle$ is recognized as an upper bound on the average *number* of distinct first-order error ligations (FLE) which may be facilitated by a randomly observed hybridization error (HE). Note: $0 \leq \chi \leq 2$.

A related quantity, the average *probability* that a randomly observed hybridization occupies a state which facilitates some first-order error ligation with an adjacent planned hybrid is estimated by:

$$p(FLE, HE) \approx \frac{\langle N_e^{fle} \rangle}{N_{max}} \xi, \quad (9)$$

where $N_{max} = 2$ is the maximum number of first-order ligation errors which any hybrid may facilitate. Since terms in the numerator of (9) are nonzero for any $ijkl$ which describes an error hybridization ($\delta_{ijk} = 1$), and has nonzero probability of first-order error ligation with an adjacent expected hybrid ($Z^{ijk} \sum_l f(m_l) \neq 0$), $\langle N_e^{ijk} \rangle / N_{max}$ is identified to be the conditional probability, $p(FLE|HE)$.

5 Results

A Java package, NucleicPark, was developed to estimate ξ and χ , assuming a Watson-Crick, statistical zipper model of duplex formation [10,30]. χ was calculated for the following encodings of Adleman's original experiment: (a) the

original set [31], (b) a set constructed from the Modified DeBruijn sequence listed in [3], (c,d) sets with good, bad modified-Hamming properties [5], (e,f) sets with good, bad stringency properties [32], and (g) a set with minimal ξ [10]. Results for each encoding, for T_{rx} from 5°C to an estimated low dissociation limit, T_{rx}^{max} , are shown in Fig. 2, along with the mean χ of a population of 1000 random encodings. An uncertainty in ΔG° was calculated for each accessible duplex configuration as suggested in [12]. The resulting fractional uncertainty in $\log \chi$, estimated by standard error propagation, was $\leq 1\%$ for all values. In

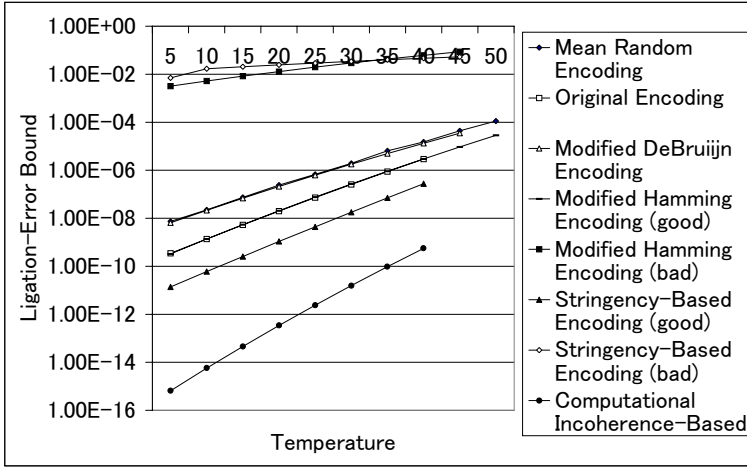


Fig. 2. $\log \chi$ vs. T_{rx} for encoding sets (a-g), reported from 5°C to an estimated T_{rx}^{max} . All values assume a Watson-Crick, statistical zipper model of duplex formation.

order to facilitate a comparison between ξ and χ , values of $\log \xi$ [10], $\log \chi$, and $p(FLE|HE)$ at 25°C for sets (a-g) are listed in Table 1. For purposes of standardization, ξ and χ was also assessed for 1000 randomly generated encodings. Both quantities were observed to follow a roughly lognormal distribution, characterized by the listed mean and standard deviations. A bound on the total initial concentration of first order ligatable substrates C_0^{fle} was also estimated using the relation, $C_0^{fle} \approx N_{total}\chi/A_vV_{rx}$, where A_v is Avagadro's number, the reaction volume, V_{rx} was taken to be $100\mu\text{L}$, and a total of 50 pmoles of each ssDNA species was assumed, for a total initial ssDNA pool of size $6.6E14$ strands. N_{total} , the total number of hybridizations, was assumed to roughly equal this number in the low dissociation limit.

Table 1. $\log_{10} \xi$, $\log_{10} \chi$, $p(FLE|HE)$, and C_0^{fle} at 25°C for encoding sets (a-g).

Encoding	$\log \xi$	$\log \chi$	$p(FLE HE)$	C_0^{fle}
Mean Random	-4.85 ± 2.26	-6.62 ± 1.48	0.085	2.64E-12 M
(a) Original	-6.01	-7.14	0.037	7.97E-13 M
(b) DeBruijn	-5.89	-6.21	0.243	6.78E-12 M
(c) Hamming (good)	-6.06	-7.12	0.0439	8.34E-13 M
(d) Hamming (bad)	-2.00	-1.97	0.985	1.18E-07 M
(e) Stringency (good)	-6.98	-8.36	0.0211	4.80E-14 M
(f) Stringency (bad)	-1.84	-1.53	1.00	3.25E-07 M
(g) ξ -Based	-10.82	-11.62	0.080	2.64E-17 M

6 Discussion

Interpreted as a measure of annealing-ligation fidelity, χ is a zero-dissociation estimate of the average propensity of a hybridized DNA mixture to generate substrates for a theoretical *ultrapotent* DNA ligase enzyme, which displays zero ability to discriminate versus partially mismatched dsDNA substrate. χ thus expresses an upper bound on the annealing-ligation error potential of a given encoding set, relative to the performance of actual DNA ligases. Because of the fragmentary, qualitative nature of current information regarding DNA ligase fidelity and processivity, the derivation of a tighter theoretical expression for a bound does not appear to be presently feasible.

The bounding nature of χ complicates absolute comparisons of the ligation fidelities of encoding sets (a-g). It would appear, however, that models of fidelity which rely strictly on DNA sequence comparison are less capable of ensuring minimal occupancy of the types of configurations which are predicted to facilitate both ligation and hybridization errors. This result suggests the necessity of including additional information regarding the specific free energies and/or occupancies of accessible duplex states. On the other hand, the actual behavior of sets (a-g) in the presence of real DNA ligases would probably, but not necessarily, correlate with the bounding behavior indicated by χ . Absolute orderings could change with the DNA ligase actually used, for instance.

An additional result is that the probability that a randomly observed hybrid is a hybridization error, and that it facilitates a first order ligation error, are not independent. Consider the probability $p(FLE, HE)$ of observing a configuration which generates both types of error. Independence implies $p(FLE, HE) = p(FLE)p(HE)$. Expression (9), however, indicates the contrary result,

$$p(FLE, HE) = \frac{\langle N_e^{fle} \rangle}{N_{max}} \xi = p(FLE|HE)p(HE). \quad (10)$$

Results in Table 1 indicate that the coupling factor $p(FLE|HE)$ is in general, poorly correlated with ξ . This result supports the intuitive notion that developing encodings with minimal biostep error-coupling might be additional means of

increasing reliability. In addition, this result suggests that attempts to construct a model of overall fidelity from independent considerations of biostep fidelity are poorly motivated, at least in the case of annealing-ligation.

Estimation of ξ and χ requires explicit calculation of the equilibrium constants of the various duplex species. This requires the use of various approximations. Values listed were computed using a classical statistical zipper model of duplex formation. As a result, values listed properly refer to the probability of observing a Watson-Crick-complementary error mode of hybridization, and should therefore be recognized to provide only an tentative picture of the overall error occupancy for the mixtures examined. In particular, there are theoretical grounds to suspect that the standard statistical zipper model may be inadequate. The validity of the model is typically justified for short DNAs by noting the large multiplicative penalty ($\sigma \approx 4.5 \times 10^{-5}$ [29]) assigned to configurations containing an internal loop, due to the loss of a stacking interaction at each end. In practice, a statistical zipper model is usually considered to be adequate for quasirandom DNAs of length less than approximately 100 base pairs, based on good agreement with experimentally derived DNA melting curves [33].

In addition to neglecting configurations containing internal loops, however, the staggered zipper model also neglects configurations containing bulges, as well as single and tandem mismatches. For the case of the melting of an ensemble of matched strands, configurations containing bulges are generated primarily through strand slippage into a stable alternative configuration, a situation which is relatively unlikely to occur during the melting of quasirandom DNAs. For applications other than melting, however, assuming the broad negligibility of bulged configurations appears to be less justifiable, due to the smaller size penalty for a single base bulge (roughly $\sigma^{1/2}$) [34]. In any case, calculated values of ξ and χ , while incomplete in the sense that a statistical zipper model was assumed, clearly indicate that the coupling factor, $\langle N_e^{fle} \rangle$ is strongly encoding dependent, and in general, much different than unity, a result which seems unlikely to be altered by utilizing a more precise model of duplex formation.

7 Conclusion and Further Work

Within the framework of a simple statistical thermodynamic model, the nature of the error coupling of successive annealing and ligation biosteps has been discussed for the limit of zero dissociation. Based on the fragmentary nature of the literature on ligase fidelity, the use of a DNA ligase of ideal infidelity, with behavior expected to bound that of actual DNA ligases, was assumed. Results indicate that models of fidelity which rely solely on DNA sequence comparison are less capable of ensuring minimal occupancy of the types of configurations which are predicted to facilitate both ligation and hybridization errors. This result suggests the necessity of including additional information regarding the specific free energies and/or occupancies of accessible duplex states. In addition, the presence of a coupling factor in the derived expression for overall fidelity

is evidence that attempts to construct analytical models of overall fidelity from independent considerations of bistep fidelity are poorly motivated.

The expressions presented for χ and ξ are independent of the model of duplex formation used to perform calculations. Listed values of ξ and χ , however, assume a statistical zipper model. Current work is focused on implementing an improved model of duplex formation. Based on observations of bulged RNAs [35], the statistical penalty applied to configurations containing a multiple-base bulge averages $\approx 4.0 \times 10^{-5}$ (roughly σ) for a two-base bulge, and increases with loop size. This suggests the use of a modified statistical zipper model, which neglects configurations with internal loops, multiple-base bulges, and/or multiple bulges, but includes configurations with both single internal mismatches and single one base bulges. Such a model maintains the polynomial size of the configurational subspace while implementing a more uniformly realistic model of duplex formation. Preliminary results indicate that configurations with internal mismatches or a single base bulge may contribute significantly to both ξ and χ .

References

1. L. M. Adleman, "Molecular Computation of Solutions to Hard Combinatorial Problems", *Science* **266**, 1021 (1994).
2. P. D. Kaplan, G. Cecchi, A. Libchaber, "DNA Based Molecular Computation: Template-Template Interactions in PCR", *DNA Based Computers III, Princeton University, 1999*, DIMACS Proc. Series (American Mathematical Society, Providence, RI, 1999).
3. W. D. Smith, "DNA Computers in Vitro and Vivo", in R. J. Lipton, E. B. Baum, editors, *DNA Based Computers*, (American Mathematical Society, Providence, RI, 1996), 121.
4. M. Amos, A. Gibbons, D. Hodgson, "Error-Resistant Implementation of DNA Computations", in, L. F. Landweber and E. B. Baum, editors, *DNA Based Computers II*, (American Mathematical Society, Providence, Rhode Island, 1999), 151.
5. R. Deaton, M. Garzon, R. E. Murphy, J. A. Rose, D. R. Franceschetti, S. E. Stevens, Jr., "Reliability and Efficiency of a DNA-Based Computation", *Physical Review Letters*, **80**, 417 (1998).
6. K. D. James, A. R. Boles, D. Henckel, A. D. Ellington, "The Fidelity of Template-Directed Oligonucleotide Ligation and its Relevance to DNA Computation", *Nucleic Acids Research*, **26**, 5203 (1998).
7. Y. Aoi, T. Yoshinobu, K. Tanizawa, H. Iwasaki, "Ligation Errors in DNA Computing", *Biosystems* **52**, 181 (1999).
8. T. Brown, D. J. S. Brown, "Purification of Synthetic DNA", *Methods in Enzymology*, **211**, 20 (1992).
9. H. Echols and M. F. Goodman, "Fidelity Mechanisms in DNA Replication", *Annu. Rev. Biochem.* **60**, 477 (1991).
10. J. A. Rose, R. J. Deaton, D. R. Franceschetti, M. Garzon, S. E. Stevens, Jr., "A Statistical Mechanical Treatment of Error in the Annealing Bistep of DNA Computation", in W. Banzhaf, et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference, Volume 2*, (Morgan Kaufmann, San Francisco, 1999), 1829.

11. C. R. Cantor, P. R. Schimmel, *Biophysical Chemistry, Part III: The Behavior of Biological Macromolecules* (Freeman, New York, 1980).
12. H. T. Allawi, J. SantaLucia, Jr., "Thermodynamics and NMR of Internal GT Mismatches in DNA", *Biochemistry* **36**, 10581-10594 (1997).
13. I. R. Lehman, "DNA Ligase: Structure, Mechanism, and Function", *Science*, **186**, 790 (1974).
14. V. Sgaramella and H. C. Khorana, "A Further Study of the T4 Ligase-catalyzed Joining of DNA at Base-paired Ends.", *J. Mol. Biol.* **72**, 493 (1972).
15. N. P. Higgins and N. R. Cozzarelli, "DNA-joining Enzymes: a review.", *Methods in Enzymology*, **68** 50 (1979).
16. C. Goffin, V. Bailly, W. G. Verly, "Nicks 3' or 5' to AP Sites or to Mismatched Bases, and One-nucleotide Gaps can be Sealed by T4 DNA Ligase", *Nucleic Acids Research* **21**, 8755 (1987).
17. K. Harada and L. E. Orgel, "Unexpected Substrate Specificity of T4 DNA Ligase Revealed by *in vitro* Selection", *Nucleic Acids Research* **21**, 2287 (1993).
18. U. Landegren, R. Kaiser, J. Sanders, L. Hood, "A Ligase-Mediated Gene Detection Technique", *Science* **241**, 1077 (1988).
19. R. Wiaderdiewicz and A. Ruiz-Carrillo, "Mismatch and Blunt to Protruding-end Joining by DNA Ligases", *Nucleic Acids Research* **15**, 7831 (1987).
20. D. Y. Wu and R. B. Wallace, "Specificity of the Nick-Closing Activity of Bacteriophage T4 DNA Ligase", *Gene* **76** 245 (1989).
21. S. Shuman, "Vaccinia Virus DNA Ligase: Specificity, Fidelity, and Inhibition", *Biochemistry* **34**, 16138 (1995).
22. F. Barany, "Genetic Disease Detection and DNA Amplification using Cloned Thermostable Ligase", *Proc. Natl. Acad. Sci.* **88**, 189 (1991).
23. J. Luo, D. E. Bergstrom, F. Barany, "Improving the Fidelity of *Thermus thermophilus* DNA Ligase", *Nucleic Acids Research* **24**, 3071 (1996).
24. C. E. Pritchard and E. M. Southern, "Effects of Base Mismatches on Joining of Short Oligonucleotides by DNA Ligases", *Nucleic Acids Research* **25**, 3403 (1997).
25. J. N. Housby and E. M. Southern, "Fidelity of DNA Ligation: a Novel Experimental Approach Based on the Polymerization of Libraries of Oligonucleotides", *Nucleic Acids Research* **26**, 4259 (1998).
26. D. D. Lasko, A. E. Tomkinson, T. Lindahl, "Eukaryotic DNA Ligases", *Mutation Research* **236**, 277 (1990).
27. A. E. Tomkinson and Z. B. Mackey, "Structure and Function of Mammalian DNA Ligases", *Mutation Research* **407**, 1 (1998).
28. A. E. Tomkinson, N. J. Tappe, E. C. Friedberg, "DNA Ligase I from *Saccharomyces cerevisiae*: Physical and Biochemical Characterization of the CDC9 Gene Product", *Biochemistry* **31**, 11762 (1992).
29. A. S. Benight, J. M. Schurr, P. F. Flynn, B. R. Reid, D. E. Wemmer, "Melting of a Self-Complementary DNA Minicircle", *J Mol Biol* **200**, 377 (1988).
30. M. Garzon, R. J. Deaton, J. A. Rose, D. R. Franceschetti, "Soft Molecular Computing", in E. Winfree and D. Gifford, editors, "Preliminary Proceedings of the Fifth International Meeting on DNA Based Computers", Massachusetts Institute of Technology, (American Mathematical Society, Providence, Rhode Island, 1999), 89.
31. L. M. Adleman, personal communication (1999).
32. B-T Zhang, S-Y Shin, "Molecular Algorithms for Efficient and Reliable DNA Computing", *Proceedings of the Third Annual Genetic Programming Conference, University of Wisconsin at Madison, 1998*, (Morgan Kaufman, San Francisco, 1998), 735.

33. A. S. Benight, R. M. Wartell, D. K. Howell, "Theory agrees with experimental thermal denaturation of short DNA restriction fragments", *Nature* **289**, 203 (1981).
34. J. Zhu, R. M. Wartell, "The Effect of Base Sequence on the Stability of RNA and DNA Single Base Bulges", *Biochemistry* **38**, 15986 (1999).
35. C. Longfellow, R. Kierzek, D. Turner, "Thermodynamic and Spectroscopic Study of Bulge Loops in Oligoribonucleotides", *Biochemistry* **29**, 278 (1990).

DNA Implementation of a Royal Road Fitness Evaluation

Elizabeth Goode*, David Harlan Wood**, and Junghuei Chen**

University of Delaware, Newark DE 19716, USA

Abstract. A model for DNA implementation of Royal Road evolutionary computations is presented. An encoding for a Royal Road problem is presented. Experimental results utilizing 2-d denaturing gradient gel electrophoresis (2-d DGGE) and polyacrylamide gel electrophoresis (PAGE) for separation by fitness in this sample Royal Road problem are shown. Suggestions for possible use of the MutS and MutY proteins as tools for separation by fitness are given. Plans for future experiments and implementation are discussed.

1 Introduction

Evolution can be viewed as the dynamical change which occurs within a population as generations of individuals are exposed to selection criteria and then allowed to reproduce. The forces of selection change the genotypic makeup of the population by removing individuals which do not meet some fitness standard, while reproduction introduces change through mutation and/or genetic recombination. The notion of evolution as a process which powers the dynamics of a system has been applied in both computing and molecular biology. These applications have given rise to the fields of study known as "evolutionary computation" and "*in vitro* evolution."

The scientific community has recently taken great interest in biomolecular models of computation. In particular, Leonard Adleman's seminal 1994 work [1] inspired a surge of research focused on exploring the possibility of using DNA or other biomolecules to solve mathematical problems which are computationally hard [2,5,17,19,35]. In fact, the research community has been working to demonstrate that it is possible to use biomolecules to perform computations which have been previously impossible using conventional silicon computers. In light of such developments, a DNA computation which can be performed *in vitro* and for which there is no theoretical or experimental barrier to large scale implementation is clearly of interest. From the beginning of DNA based computing to the present there have been calls [6,26,27,35] to carry out evolutionary computations using genetic materials in the laboratory. Our model for DNA implementation of evolutionary computations addresses these issues.

* Supported by NSF Grants No. 9805703 and No. 9980092

** Partially supported by NSF Grant No. 9980092 and DARPA/NSF Grant No. 9725021.

1.1 The Royal Road

The Royal Road problems are a class of evolutionary computations which were initiated by Mitchell, Forrest and Holland [23]. Recent research of van Nimwegen *et.al.* [28] emphasized the population dynamics of various Royal Road fitness functions. Royal Road problems often exhibit “evolutionary stasis,” time periods when essentially no change takes place in population fitness. Stasis is one of the most interesting features of Royal Road because it is also frequently observed in both natural evolution and in evolutionary computation. In fact, van Nimwegen *et.al.* draw attention to Royal Road as a model of natural evolution. The theoretical results of van Nimwegen *et.al.* predict that in some situations stasis lasts for only a relatively few generations. However, their computer simulations do not support their theoretical results on the duration of stasis. They identify the lack of sufficient genetic variation in their populations as the likely cause of the observed discrepancy between theory and computation. In their computations, genetic variation is most likely limited because most of the highly fit individuals are presumably the descendents of a single favorably mutated individual.

Royal Road using DNA could test the theoretical predictions of stasis duration due to van Nimwegen *et.al.* Population genetic variation could be maintained in DNA’s huge populations because favorable mutations would be proportionately less rare. By implementing Royal Road problems using DNA, one would be able to use populations many orders of magnitude larger than the populations available using conventional computers. Huge DNA storage capacity permits exploring populations with much greater genetic diversity than populations available to van Nimwegen *et.al.* Their largest populations contained 10^4 individuals using about 10^5 bytes. Meanwhile, one microgram of DNA (typical in experiments) corresponds to more bytes of information than the 1995 production of computer hard disks [?]. However, with our present laboratory techniques, we would be limited to demonstrations involving very few generations. This is because each generation would take about one day in the laboratory regardless of the populations size.

Thus, using huge populations encoded in DNA and relatively few generations, we would be able to test precisely those theoretical predictions van Nimwegen *et.al.* were not able to verify due to restricted population sizes.

The work in this paper focuses solely on fitness-based separation of individuals for a Royal Road problem. Fitness-based separation of individuals is a crucial step in our DNA model for simulating Royal Road computations. The important fact for the biomolecular computing community is that our separation technique utilizes the separation capability of 2-d denaturing gradient gel electrophoresis (2-d DGGE) as well as non-gradient gel electrophoresis (PAGE). We also discuss the theoretical basis for using mismatch-binding proteins in combination with gel shift assays for separation by fitness. We describe our progress in developing separation techniques for our DNA implementation of evolutionary algorithms in detail.

Populations of individuals separated by fitness via the 2-d DGGE, PAGE and protein-DNA gel shift assays can then be further evolved by other labora-

tory procedures which are available today. Presently we focus our research on determining the feasibility of implementing the separation-by-fitness step, and describe our experimental results. We anticipate exploring the latter steps in our future research. We believe our DNA implementation of the Royal Road should be of interest to both the biomolecular computing community and the evolutionary computation community because of the potential of computing with very large populations, and because the use of the 2-d DGGE, PAGE and other techniques may suggest other applications or techniques hitherto unexplored.

2 Definitions and Examples

2.1 Evolutionary Algorithms

We begin with a brief overview of the general notion of an evolutionary algorithm, and refer the reader to [415] for further details as required. We focus particularly on the notion of the fitness function within an evolutionary algorithm.

An evolutionary algorithm begins with a population (possibly random) which is subjected to iterated cycles of selection and reproduction. Selection is by some defined fitness criteria, *i.e.*, the fitness function. Individuals are evaluated according to the fitness function, and sufficiently fit individuals are selected. Selected individuals are allowed to reproduce the next generation of individuals according to some reproduction strategy which may include mutation and crossover (genetic recombination). Most common may be the selection and reproduction strategies which allow reproduction as a weighted probabilistic function of the relative fitness of each individual.

An example is the MaxOnes problem. The MaxOnes computation begins with a random set of individual bitstrings of zeroes and ones, each of length n . The desired outcome is a 'perfect' individual bitstring of all ones. For a given initial population size, the goal is to generate such perfect individuals. The fitness function usually chosen is a simple evaluation based on the number of ones in a given individual's sequence.

2.2 The Royal Road Fitness Function

The Royal Road fitness function is a generalization of the MaxOnes fitness function. Rather than simple zero/one bitstrings in which the count of ones determines the fitness of the individual, the population of individuals in the Royal Road are strings which contain discrete *blocks* which are subsequences of bits. Each block is evaluated for fitness. Most usually, the requirement is that each block consist of all ones, but other block requirements are possible.

Each block in a given individual bitstring which satisfies its predefined block fitness criterion contributes to the fitness rating of that individual. A block in which there are any deviations from the required specification fails to contribute to the individual's overall fitness. The sum of the block contributions constitutes the total fitness for the bitstring. Most often, blocks are assigned fitness 1 if

they are perfect, and fitness 0 otherwise. Selection is a function of the fitness. Reproduction may be according to any desired paradigm, although it is generally independent of fitness.

2.3 DNA: Some Biochemistry

DNA, or deoxyribonucleic acid, is the genetic material of all living things. DNA can be found in both single-stranded form (ssDNA) and double stranded form (dsDNA) in nature, and ssDNA can be manufactured synthetically. Each DNA single strand consists of an ordered sequence of four distinct *bases*: adenine, guanine, cytosine and thymine. These bases are abbreviated as A, G, C and T, respectively. The bases in a single strand of DNA are held together by covalent bonds.

DNA strands have a direction, customarily denoted as 5' to 3', as a consequence of the way in which the bases covalently bond to one another. The structure of dsDNA is a double helix of two single strands. Hydrogen bonds naturally form between the paired bases A and T and between C and G. These are called *complementary* bases. Two single strands having sequences of complementary bases are called *complementary strands*, or just *complements*. The two complementary single strands of DNA in a double helix have opposite directions.

The hydrogen bonding process by which complementary single DNA strands join together to form dsDNA is called *hybridization* or *annealing*. A double strand of DNA can be separated into single strands by heating (melting), a process called *dehybridization*. The temperature at which a double stranded DNA dehybridizes is referred to as its *melting temperature*. Different sequences of dsDNA have different melting temperatures.

It is possible for two single strands which are not perfectly complementary to one another to bond together into a double strand, although the structure of that strand is not always a perfect double helix. The annealed product of a single strand of DNA with its perfect complementary strand will melt at higher temperature than the annealed product of the same single stranded DNA with another strand which is not its perfect complement. We shall exploit this fact for our DNA implementation of the Royal Road problem.

3 Motivation – Why DNA on the Royal Road?

Our goal is to demonstrate experimentally that an instance of the Royal Road problem can be implemented using DNA. We focus here on implementing the selection step of an instance of the Royal Road problem. We have chosen to implement the Royal Road fitness function using DNA for several compelling reasons.

First, van NimWegen *et.al.* [30,29] examined the population dynamics in instances of the Royal Road problem involving populations of various sizes. The largest population size they treated was on the order of 10^4 . In our DNA implementation model we can use populations of size 10^{12} or larger. Our DNA

computations therefore have the potential of generating previously unobtained information about the dynamics of Royal Road computations.

Second, we chose the Royal Road problem because of the feasibility of the necessary laboratory steps required for DNA implementation. The Royal Road is a generalization of the MaxOnes algorithm, and Chen *et.al.* demonstrated the DNA implementation of the fitness evaluation step of the MaxOnes algorithm [9]. We attempt to apply what has been learned in the previous implementation to a new phase of DNA implementation.

We anticipate using the tools we develop as a first step in the development of separation tools which can be applied to sample spaces which reside in search spaces that are of large size relative to the sample population. We also anticipate expanding the range of population size as we develop new, possibly automated, laboratory techniques. Because of the enormous storage capacity of DNA, the potential gain in computing evolutionary algorithms using DNA rather than silicon is unprecedented. We expect that results we obtain concerning the Royal Road applied to very large populations will be of interest to the DNA computing community as well as the evolutionary computation community.

4 The Preliminary Example for Royal Road Fitness-Proportional Selection

Let $A = \{C, T, G\}$ be our working set of symbols. The block alphabet is $B = \{C, T\}$. The population of interest is a set of bitstrings of length 88 written over A , each containing 2 blocks written over B of length 6 in bit positions 25-30 and 57-62. We consider individuals to be distinguishable only by the content of their blocks. Therefore the population contains at most 2^{12} individuals. The Royal Road fitness function for the preliminary example assigns fitness 1 for each perfect block containing all T s. Thus a perfect individual contains only T in each of its blocks, and has fitness 2. An individual which has one perfect block of all T , and one block containing at least one C is assigned fitness $1+0=1$. An individual which has at least one C in each of its blocks has fitness $0+0=0$. Individuals with high fitness are likely to be selected for reproduction.

There are a number of issues which must be treated in order to implement the preliminary example using DNA. The individuals, once encoded in DNA, must be physically separable by fitness. We have evidence which supports our thesis that 2-d DGGE in combination with PAGE implements this fitness function. We anticipate doing selection over the entire population of one generation in one day. While the treatment of thousands of generations may not be possible without robotics, it will, we believe, be possible to treat populations of size 10^{16} or greater using the laboratory techniques presently available. It is not practical to use conventional computers for populations of this size. In the next section we discuss the specifics of our DNA implementation design for the preliminary example.

5 The Experimental Design

5.1 Perpendicular 2-d DGGE

The motivation of our work is to demonstrate that separation by fitness for the preliminary example (given above) of the Royal Road fitness function can be performed using a combination of 2-d denaturing gradient gel electrophoresis (2-d DGGE) and polyacrylamide gel electrophoresis (PAGE). We also suggest that mismatch-binding proteins MutS and MutY may be useful for separation by fitness.

Denaturing gradient gel electrophoresis is a method by which single base changes in DNA strands may be identified. This technique, first introduced by Fischer and Lerman [12], involves exposing dsDNA to an environment containing a gradient of denaturant concentration. The dsDNA is moved through the gradient gel environment by electrophoresis. Partial dehybridization of dsDNA in a denaturing environment reduces the mobility of the DNA through polyacrylamide gel. Since melting temperatures of dsDNA are sequence specific, the different melting temperatures of different sequences yield differences between the movement of those sequences through a denaturing gradient gel, even if those sequences are the same length.

We used polyacrylamide perpendicular denaturing gradient gels to perform our selection by fitness. The perpendicular gradient gel has a chemical gradient along its x-axis dimension, and the electric field is applied in the y-axis direction. Samples are loaded across the x-axis, and run downward in the vertical direction, so that each vertical line of sample passes through a particular denaturing environment for the duration of the electrophoresis. Since many vertical lines of sample pass through the x-axis, and each vertical slice of gel has a denaturing concentration which is slightly different than that of every other vertical slice, a large quantity of information about the sample can be obtained during a single electrophoretic run.

The information gathered by 2-d DGGE can then be used to determine an optimal denaturing gradient at which separation occurs between candidates of different fitness. Separation results are then verified with non-gradient PAGE gels.

5.2 The Candidate Individuals

The 88-bit long individuals, or *candidates* of the preliminary example are encoded as single-stranded DNA (ssDNA) consisting of 88 bases each. Each individual strand consists of 5 concatenated sequences of cytosine (C), guanine (G) and thymine (T). No adenine (A) was used in the encoding of the candidates in the population. The candidates used are all concatenates of the following five sequences: Clamp1, Block1, Clamp2, Block2, and Clamp3. The clamps are distinct, but constant for all candidates, and have lengths 24, 26 and 26, respectively. The three clamp sequences are G-C rich regions. The blocks have length 6, and contain a mixture of C and T, varying among different candidates. Blocks

contain only T and C, and perfect blocks consist of all T. The 'perfect' candidate therefore has only T in Block1 and Block2.

The candidate strands can theoretically be divided into equivalence classes by fitness. Those candidates having at least one C in each of Block1 and Block2 have fitness 0. Those candidates having one perfect block containing only T, and one imperfect block containing at least one C are assigned fitness 1. The perfect candidate has only T in both Block1 and Block2, and is assigned fitness 2. Since the clamps are constant for all individuals, there is only one sequence associated with a perfect individual.

We have set out to show experimentally that we can physically divide our candidate strands into equivalence classes. In order to achieve this separation, we anneal the various 'imperfect' candidates to the complement of the perfect candidate, called the *Target*. The Target is a necessary element for separation of candidate sequences by fitness. Those candidates which have fit blocks are predicted to anneal more perfectly to the Target than those candidates which have unfit blocks. This variation in hybridization, and the resulting variation in melting temperatures of the annealed products, should be separable via 2-d DGGE.

The perfect candidate strand, called Candidate Perfect, is a strand having fitness 2, since both blocks contain only T. Candidate Perfect is the ssDNA with the following sequence (written 5' to 3') in which the clamps and blocks are separated by spaces for easy reading:

5' -- GGGCGGCCTCGCCTCCCTGCTGG TTTTT CTTCTCCCTCTGTCTGGGCTCGCGTT
TTTTTT TTGTTGCTTCGTTTGTCTTCCGTCC -- 3'

Candidate 2.1 is a candidate having fitness 1. Candidate 2.1 has a perfect Block1 of all T's, and Block2 contains 1 mismatch base C rather than T. The sequence for Candidate 2.1 is given by:

5' -- GGGCGGCCTCGCCTCCCTGCTGG TTTTT CTTCTCCCTCTGTCTGGGCTCGCGTT
CTTTTT TTGTTGCTTCGTTTGTCTTCCGTCC -- 3'

Candidate 2.6 is another candidate having fitness 1. Candidate 2.6 has a perfect Block1, and Block2 contains all C's. Notice that Block2 of Candidate 2.6 is as mismatched from the Block2 of Candidate-Perf as is possible:

5' -- GGGCGGCCTCGCCTCCCTGCTGG TTTTT CTTCTCCCTCTGTCTGGGCTCGCGTT
CCCCC TTGTTGCTTCGTTTGTCTTCCGTCC -- 3'

Candidate 1.6-2.6 is a candidate having fitness 0. In Candidate 1.6-2.6, both Block1 and Block2 contain all C's :

5' -- GGGCGGCCTCGCCTCCCTGCTGG CCCCC CTTCTCCCTCTGTCTGGGCTCGCGTT
CCCCC TTGTTGCTTCGTTTGTCTTCCGTCC -- 3'

The Target strand, which is the exact complement of Candidate Perfect, has the following sequence:

5' -- CGACGGAAGGACAAACGAAGCAACAA AAAAA AACGCGAGCCCGACAGAGGGAGAAGG
AAAAA CCAGCAGGGGAGGCGAGGCCGCC --3'

5.3 Separation by Fitness Using 2-d DGGE and PAGE

Separation by fitness must be demonstrable using laboratory techniques. We show that 2-d DGGE in combination with PAGE allows separation of a subset of our candidate strands by fitness class.

In theory, different candidate strands annealed to the Target strand should run differently according to their fitness. Those candidates having blocks which perfectly anneal to their corresponding complement sequences in the Target strand are predicted to run more quickly through a gel than candidate strands which have one or more occurrences of a C in one or both blocks. In theory, each of the three fitness equivalence classes of candidate strands as defined by our instance of the Royal Road fitness function should be distinguishable by 2-d DGGE. Verification of separability is performed using the non-gradient PAGE technique.

Experimental runs of the candidates in fitness class 1 and the perfect candidate having fitness 2 were performed using 2-d DGGE. Both the Candidate 2.1/Target and Candidate 2.6/Target annealed products are shown to run more slowly through a 15% acrylamide gel (0%-60% denaturing gradient) than the Candidate Perfect/Target annealed product. Our results using these candidates support our thesis that these techniques can be used to implement selection by fitness for this Royal Road fitness function.

Experimental runs of the candidates in fitness class 1 and the candidate of fitness 0 were also performed. The Candidate 1.6-2.6/Target annealed product is shown to run more slowly than the Candidate 2.1/Target and Candidate 2.6/Target products, as expected. Again, a 15% acrylamide gel was used, at 0% denaturing gradient.

6 Laboratory Procedure

All candidates of fitness less than 2 and the Target ssDNA molecules were obtained from Life Technologies. Candidate Perfect was generated using the polymerase chain reaction with the Target as the template strand. We used 20 base long primers obtained from Geneco in a PCR reaction containing 1.5mM $MgCl_2$ and an annealing temperature of 55°C.

The synthetically produced oligos were purified by denaturing polyacrylamide gel electrophoresis, and subjected to phenol/chloroform/isoamyl alcohol extraction, and ethanol precipitation and ethanol wash procedures. The PCR product Candidate Perfect/Target was purified with the S-400 column from Pharmacia.

In order that the candidate strands could be visualized, a portion of each sample of purified oligos was radiolabeled by kinasing with P- 32. After kinasing, oligos were cleaned up with G-50 columns from Pharmacia. The working concentration of the ssDNA was on the order of 0.1 pmol/ μ l. The dsDNA Candidate Perfect/Target sample had a working concentration of about 0.5pmol/ μ l. Samples containing approximately 1.5 pmol unlabeled Target and 0.14 pmol labeled Candidate 2.1, Candidate 2.6 or Candidate 1.6-2.6 were annealed in 1 x

TAE and 10mM Mg⁺⁺ buffer. The annealing reactions were heated to 95°C for 10 minutes, then allowed to cool to room temperature slowly over the course of 90 minutes. These annealed samples along with the labeled Candidate Perfect/Target sample were then subjected to both 2-d DGGE using the DCode DGGE System manufactured by BioRad, and PAGE. An acrylamide/bis ratio of 29:1 was used, with a final concentration of 15% acrylamide in the gels. In the 2-d gels, both urea and formamide were used to create the denaturing gradient, with the highest concentration of urea and formamide being 25% and 24/100 (vol/vol), respectively. All gels were run at 350 Volts for 4 to 5 hours at approximately 6.5°C. Imaging was performed by exposing the gels for about 12 hours in the Storage Phosphor Screen manufactured by Molecular Dynamics.

7 Results and Discussion

The fitness selection was implemented by observing the variations of movement through the 2-d denaturing gel of the different candidates annealed with the Target strand. It was expected that the CandidatePerfect, as an annealed product with the Target, would run more quickly through the gradient gel than the annealed product the Target with a candidate of fitness less than 2. This prediction was based on the fact that a candidate of fitness less than 2 has some degree of mismatch in one or both of its blocks with the corresponding complementary block portions of the Target strand. These mismatches, where C's are present rather than the perfect complement of all T's, were expected to produce an annealed product which, in some range of temperature and chemical gradient, would have fewer intact hydrogen bonds than the annealed product of the perfect compliments Target and Candidate-Perf. In particular, we expected that the imperfectly matched regions in the dsDNA formed by annealing the Target to a candidate of low fitness (fitness less than a perfect 2) would dehybridize at a lower denaturing gradient than their perfectly matched counterparts in the Candidate Perfect/Target annealed product. It was predicted that dehybridized sections or 'bubbles' in the block regions of the Target annealed with candidates of fitness less than 2 would cause these DNA molecules to move through some gel gradient more slowly than the perfectly matched dsDNA Candidate Perfect/Target strands. Further, we predicted that a candidate of fitness 0 would run more slowly than a candidate of fitness 1, and a candidate of fitness 1 would run more slowly than the perfect candidate.

Our results demonstrate that indeed we can separate annealed products containing the lower fitness candidates from the annealed product Candidate Perfect/Target, and that we can separate the candidate of fitness 0 from the fitness 1 candidates. On the 2-d gel the Candidate 2.6/Target product ran more slowly than the Candidate 2.1/Target product, and the Candidate 2.1/Target product ran more slowly than the Candidate Perf/Target product at the low denaturing end of the gradient (see Figure 1). These results were then verified using PAGE (see Figure 2). Further, with PAGE, the Candidate 1.6-2.6/Target an-

annealed product ran more slowly than any other annealed product (see Figure 3). These results are in line with our predictions.

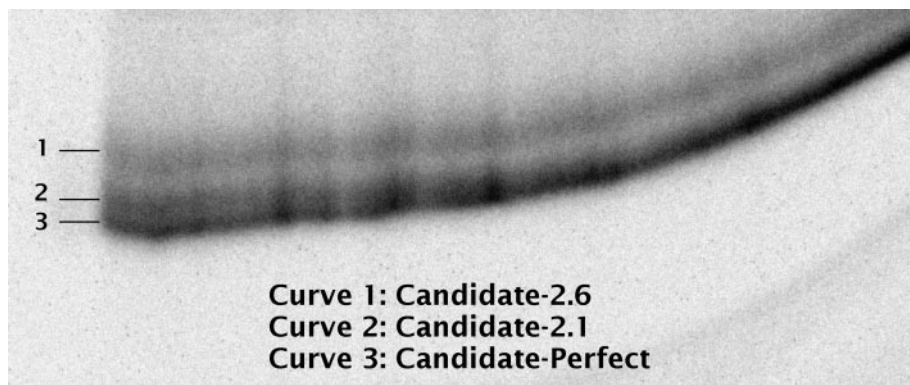


Fig. 1. Image of Candidate-Perf/Target, Candidate-2.1/Target and Target/Candidate 2.6 annealed reactions exposed to 2-d DGGE. The gel is 15% acrylamide/bis (29:1), 0%-60% denaturing gradient, run at 350 Volts for 4.5 hrs at 6.5°C. Only the P-32 labeled candidate strands are visible.

Figure 1 shows the 2-d DGGE of Candidate Perfect/Target run simultaneously with Candidate 2.1/Target and Candidate 2.6/Target. Candidate Perfect/Target, Candidate 2.1 and Candidate 2.6 are radiolabeled with P-32. Candidate Perfect is clearly distinguishable from Candidate 2.6 in the left side of the gel picture, as indicated by the labels in the figure. Candidate 2.1 runs just slightly above the Candidate Perfect in this gel. The 15% acrylamide gel (0%-60% denaturing gradient) was run at 350 Volts for 4.5 hours at 6.5°C.

In Figure 2, the identity of all strands involved in Figure 1 were verified by PAGE using separate lanes containing each annealed sample. As predicted, the annealed product Candidate 2.6/Target runs most slowly, and that Candidate 2.1/Target runs between Candidate 2.6/Target and Candidate Perfect/Target.

In Figure 3, PAGE was used to demonstrate that the Candidate 1.6-2.6/Target product runs more slowly than any of the other Candidate/Target products. Again, a 15% acrylamide gel with 0% denaturing gradient was used.

These results encourage us to believe that our model for Royal Road implementation using blocks which melt is a good predictor of the actual behavior of oligos designed to implement this problem. Further, we are encouraged to believe that future experiments with other candidates will behave in a similarly predictable manner, and that as a consequence, we shall be able to determine experimentally how to separate all of the candidates in our sample space by fitness equivalence class. Our results support our hypothesis that separation of candidates by fitness will be possible.

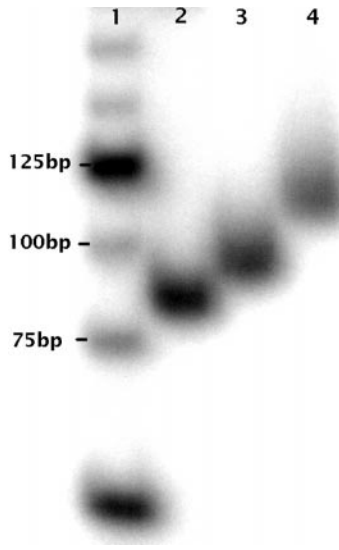


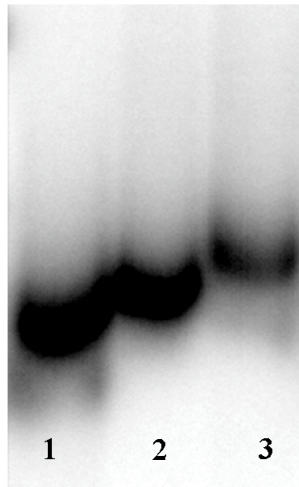
Fig. 2. Image of Candidate-Perf/Target, Candidate-2.1/Target and Target/Candidate 2.6 annealed reactions exposed to PAGE. Lane 1 contains a 25bp ladder, with bright bands at 125bp and 50bp. Lane 2 contains Candidate Perfect/Target. Lane 3 contains Candidate 2.1/Target. Lane 4 contains Candidate 2.6/Target. The gel is 15% acrylamide/bis (29:1), 0% denaturing gradient, run at 350 Volts for 4.5 hrs at 6.5°C. Only the P-32 labeled candidate strands are visible.

7.1 Separation by Fitness: Next Steps

The ability to separate fitness 1 candidates from the fitness 2 candidate is a critical test for our DNA implementation of this Royal Road algorithm. We are encouraged that we can differentiate the movement of a fitness 1 candidate from the movement of the perfect fitness 2 candidate. Considering the somewhat symmetric design of our oligos, we have good reason to believe that candidates having fitness 1 by virtue of having a similar mismatch in Block1 and a perfect sequence of T's in Block2 should also be distinguishable from Candidate Perfect. While we have yet to test all possible candidates of fitness 1 in comparison with Candidate Perfect, we reason to believe that candidates having mismatches in either Block1 and Block2 will be at least as easily distinguishable from the Candidate Perfect as is Candidate 2.1, which has only one mismatch in one block. Since we have results which show that separation between Candidate 2.1 and Candidate Perfect is possible, we are encouraged to believe that the separation of individuals having more than one mismatched base will be possible.

We also need to be able to distinguish candidates having fitness 0 from those of fitness 1, and candidates having fitness 0 from Candidate Perfect of fitness 2. These cases shall all be explored in future work as we continue to determine the viability of our DNA implementation of this Royal Road fitness function.

**PAGE of Candidates 2.1, 2.6
and 1.6-2.6**



- 1: Candidate 2.1**
2: Candidate 2.6
3: Candidate 1.6-2.6

Fig. 3. Image of Candidate-2.1/Target, Candidate 2.6/Target and Candidate 1.6-2.6/Target annealed reactions exposed to PAGE. Lane 1 contains Candidate 2.1/Target, Lane 2 contains Candidate 2.6/Target, and Lane 3 contains Candidate 1.6-2.6/Target. The gel is 15% acrylamide/bis (29:1), 0% denaturing gradient, run at 350 Volts for 4.5 hrs at 6.5°C. Only the P-32 labeled candidate strands are visible.

8 Directions for Future Research

DNA implementation of a Royal Road fitness evaluation may be possible. We are encouraged by our results to believe that our method of separation will work for candidates in this instance of the Royal Road. Further work applied to this Royal Road problem and other evolutionary problems is necessary. In particular, we need to demonstrate that we can separate fitness equivalence classes for Royal Road fitness functions in general.

In addition to using the gel electrophoresis techniques presented here, we entertain the possibility of using mismatch-repair enzymes such as MutS and MutY, which bind to specific mismatched base-pairs. We envision encoding our blocks with mismatches which can be bound by these proteins, and then detected

by gel shift assays. We are in the preliminary stages of testing this idea, following the protocols found in [3], [7], [18], [21], [22], [25] and [?]. Such assays may prove to be useful for the detection and separation of candidates having blocks which are 'almost perfect' - *i.e.* which have a single mismatch.

In later stages of our research we will implement the selection and reproduction phases of the Royal Road problem. Once candidates have been physically separated into groups of equal fitness, many different selection criteria could be applied. For example, fitness proportional selection might be done by cutting samples from the various fitness classes, diluting/amplifying samples to a standard concentration, and then combining these samples in quantities proportional to their fitnesses.

Both crossover and mutation will be incorporated in our DNA implementation of reproduction. We will take advantage of laboratory protocols which are known to induce variable levels of mutation [11,16,20,32]. Further, DNA implementation of crossover has been demonstrated by Chen *et.al* in [9]. Since mutation is often emphasized in theoretical studies of the Royal Road problem, and crossover may be useful for certain evolutionary computations, we see the availability of these protocols as a distinct advantage for implementing our model.

Finally, larger Royal Road problems could be implemented using the basic ideas presented here. Longer oligos containing block regions which are longer could theoretically be used to encode larger problems. The same basic separation technique might be applied, as could the selection and reproduction steps discussed above. DNA simulation of evolutionary computations involving huge populations is, of course, the ultimate goal.

9 Conclusions

We have experimental results which are shed light on the question "Can 2-d DGGE and PAGE be used for separating candidates according to fitness in a Royal Road evolutionary computation?" We are encouraged by our results that such fitness-based separation will be possible, and that our clamp-block style encoding of individuals is useful for DNA implementation of a Royal Road problem. More work needs to be done, both in verifying a complete separation ability for the Royal Road fitness function chosen here, and in exploring the possible uses of 2-d DGGE as the selection tool for other evolutionary algorithms implemented with DNA. We also shall explore other tools for separation by fitness, including mismatch-binding proteins in conjunction with gel shift assays.

If any of these tools are to be truly useful for selection in DNA evolutionary algorithms involving huge populations, then we must demonstrate that we can handle problems involving populations larger than that treated in this sample problem. Refinement and expansion of our proposed techniques will be required.

In conclusion, we believe that the 2-d DGGE and PAGE separation method will be useful for implementing fitness separation for the Royal Road problem and for other evolutionary algorithms. We are encouraged to believe that we

may be able to treat populations which are much larger than can be treated by conventional computers. Further research is necessary to develop a clearer picture of how this separation method may be most useful for DNA computing.

References

1. Leonard M. Adleman, *Computing with DNA*, Scientific American **279** (1998), 54–61.
2. Leonard M. Adleman, *Molecular computation of solutions to combinatorial problems*, Science **266** (1994), 1021–1024.
3. K. G. Au, S. Clark, J. H. Miller and P. Modrich, *Escherichia coli MutY gene encodes an adenine glycosylase active on G-A mispairs*, PNAS **86** (1989), 8877–8881.
4. Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz, eds., *Handbook of Evolutionary Algorithms*, Institute of Physics Publishing, Philadelphia, 1997.
5. Dan Boneh, Christopher Dunworth, and Richard J. Lipton, *Breaking DES using a molecular computer*, Tech. Report CS-TR-489-95, Princeton University, May 1995.
6. Alan Dove, *From bits to bases: Computing with DNA*, Nature Biotechnology **16**, no. 9, (1998), 830–832.
7. I. Biswas and P. Hseih, *Identification and Characterization of a Thermostable MutS Homolog from Thermus aquaticus*, The Journal of Biological Chemistry **271**, (1996), no. 9, 5040–5048.
8. J. Chen, E. Antipov, B. Lemieux, W. Cedeno, and D.H. Wood, *DNA Computing implementing genetic algorithms*, Preliminary Proceedings DIMACS Workshop on Evolution as Computation, (L. Landweber, R. Lipton, E. Winfree and S. Freeman, eds), DIMACS, Piscataway, NJ, 1999, 39–49.
9. David Harlan Wood, Junghuei Chen, Eugene Antipov, Bertrand Lemieux, and Walter Cedeño, *In vitro selection for a OneMax DNA evolutionary computation*, DNA Based Computers V: DIMACS Workshop, DIMACS series in discrete mathematics and theoretical computer science, June 14–15, 1999, (David Gifford and Erik Winfree, eds.), American Mathematical Society, Providence, to appear.
10. A. Ausubel, R. Brent, R.E. Kingston, D.D. Moore, J.G. Seidman, J.A. Smith, and K. Struhl, *Current Protocols in Molecular Biology*, Greene Publishing Associates and Wiley-Interscience, 1994.
11. J. C. Cox, P. Rudolph, and A. D. Ellington, *Automated RNA selection*, Biotechnology Progress **14** (1998), no. 6, 845–850.
12. S. Fischer and L. Lerman, Proceedings of the National Academy of Science **80** (1983), 1579–1583.
13. Philippe Giguere and David E. Goldberg, *Population sizing for optimum sampling with genetic algorithms: A case study of the Onemax problem*, Genetic Programming 1998: Proceedings of the Third Annual Conference at Madison, WI, (John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, eds), Morgan Kaufman, San Francisco, 1998, 22–25.
14. *Searching for gene defects by denaturing gradient gel electrophoresis*, Trends in Biochemical Sciences **172** (1992), no. 3, 89–93.
15. Jörg Heitkötter and David Beasley, *The hitch-hiker's guide to evolutionary computation*, (FAQ for comp.ai.genetic). Web page at <http://alife.santafe.edu/joke/encore/www/>, September 1999.

16. A. A. Beaudry and Gerald E. Joyce, *Directed evolution of an RNA enzyme*, Science **257** (1992), 635–641.
17. Lila Kari, *DNA computing: Arrival of biological mathematics*, Math. Intelligencer **19** (1997), no. 2, 9–22.
18. Xianghong Li, Patrick M. Wright and A-Lien Lu, *The C-terminal Domain of MutY Glycosylase Determines the 7,8-Dihydro-8-oxo-guanine Specificity and Is Crucial for Mutation Avoidance*, The Journal of Biological Chemistry **275** (2000), no. 12, 8448–8455.
19. Richard J. Lipton, *DNA solution of hard computational problems*, Science **268** (1995), 542–545.
20. J. R. Lorsch and J. W. Szostak, *In vitro evolution of new ribozymes with polynucleotide kinase activity*, Nature **371** (1993), 31–36.
21. *A Novel Nucleotide Excision Repair for the Conversion of an A/G Mismatch to C/G Base Pair in E. coli*, Cell **54** (1988), 805–812.
22. A-Lien Lu and Ih-Chang Hsu, *Detection of Single DNA Base Mutations with Mismatch Repair Enzymes*, Genomics **14** (1992), 249–255.
23. Melanie Mitchell, Stephanie Forrest, and John Holland, *The royal road for genetic algorithms: Fitness landscapes and GA performance*, Proceedings of the First European Conference on Artificial Life, MIT Press/Bradford Books, Cambridge, MA, 1992.
24. Melanie Mitchell, *An Introduction to Genetic Algorithms*, MIT Press, Cambridge, MA, 1998.
25. Paul Modrich, *Mechanisms and Biological Effects of Mismatch Repair*, Annu. Rev. Genet. **25** (1991), 229–253.
26. H. Muir, *DNA reveals its talent for computing*, New Scientist **144** (1994).
27. Robert Pool, *Forget silicon, try DNA*, New Scientist **151** (1996) no. 2038, 26–31.
28. Erik van Nimwegen, James P. Crutchfield and Melanie Mitchell, *Statistical Dynamics of the Royal Road Genetic Algorithm*, Theoretical Computer Science, special issue on Evolutionary Computation, to appear (1998).
29. James P. Crutchfield and Erik van Nimwegen, *Optimizing epochal evolutionary search: Population-size independent theory*, SFI Working Paper 98-06-046, 1998, 18 pages. Paper found at URL: <http://www.santafe.edu/projects/evca/evabstracts.html#ooespsit>.
30. James P. Crutchfield and Erik van Nimwegen, *Optimizing epochal evolutionary search: Population-size dependent theory*, SFI Working Paper 98-10-090, 1998, 18 pages. Paper found at URL: <http://www.santafe.edu/projects/evca/evabstracts.html#ooespsdt>.
31. James P. Crutchfield and Erik van Nimwegen. The evolutionary unfolding of complexity. In Laura Landweber, Erik Winfree, Richard Lipton, and Stephan Freeland, editors, *Proceedings of the DIMACS Workshop on Evolution as Computation*, New York, 1999, to appear. Springer-Verlag.
32. M. Sassanfar and J. W. Szostak, *An RNA motif that binds ATP*, Nature **364** (1993), 550–553.
33. Gerhard Steger, *Thermal denaturation of double-stranded nucleic acids: Prediction of temperatures critical for gradient gel electrophoresis and polymerase chain reaction*, Nucleic Acids Research **22** (1994), no. 14, 2760–2768.
34. Willem P.C. Stemmer, *DNA shuffling by random fragmentation and reassembly: In vitro recombination for molecular evolution*, Proceedings of the National Academy of Science, U.S.A. **91** (1994), 389–391.
35. Willem P.C. Stemmer, *The evolution of molecular computation*, Science **270** (1995), 1510–1510.

36. Willem P.C. Stemmer, *Sexual PCR and Assembly PCR*, The Encyclopedia of Molecular Biology and Molecular Medicine, (Robert Meyers, ed), VCH, New York, 1996, 447–457.
37. D.H. Wood, J. Chen, E. Antipov, W. Cedeno, and B. Lemieux, *A DNA implementation of the Max 1s problem*, GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference, July 1999, Orlando, Florida, (W. Banzhaf, A.E. Eiben, M. Garzon, V. Honavar, M. Jakiela, and R.E. Smith, eds), Morgan Kaufman, San Francisco, 1999, 1835–1842.

Steady Flow Micro-Reactor Module for Pipelined DNA Computations

John S. McCaskill, Robert Penchovsky, Marlies Gohlke, Jörg Ackermann, and
Thomas Rücker

GMD-National Research Center for Information Technology
Schloß Birlinghoven, St. Augustin 53754, Germany
Email: mccaskill@gmd.de

Abstract. Microflow reactors provide a means of implementing DNA Computing as a whole, not just individual steps. Contrary to surface based DNA Chips[1], microflow reactors with active components in closed flow systems can be used to integrate complete DNA computations[2]. Microreactors allow complicated flow topologies to be realized which can implement a dataflow-like architecture for the processing of DNA. A technologically feasible scalable approach with many reaction chambers however requires constant hydrodynamic flows. In this work, the experimental construction of a basic constant flow module for DNA processing in such a context is addressed. Limited diffusional exchange in parallel flows is used to establish spatio-temporal segregation of reaction conditions which can be crossed by magnetic beads without barriers. As previously outlined[2], linked up with an optical programming technology, this will enable DNA selection to be programmed and complex population selection to be performed. The basic first experimental step in the realization of this program is described here: the establishment of a stable hydrodynamic flow pattern which is scalable to many reactors in parallel and the demonstration of a scalable and synchronous clocking of magnetic bead-based processing. First results with fluorescently-labeled DNA transfer will also be presented at the conference. The way in which this module may be integrated to solve the maximal clique problem has been proposed elsewhere[2].

1. Introduction

Integrating DNA computing to the point where complex computations can be performed routinely is a major conceptual and technological challenge. Whereas early experiments with DNA processing involved manual processing steps[3], the scalability of DNA computing to significant problem sizes requires new progress in biotechnological integration. Suitably matched architectures for DNA processing must be developed which take advantage of the integration potential of such technology. Two major trends in such Integration may be discerned: DNA Chip technology (as in [1]) on the one hand and microflow systems (as employed in μ TAS[4]) involving fluid flow in sealed channel networks on the other. In previous work[5], we have described the application of microflow reactors to spatially structured molecular evolution experiments with DNA and RNA. Completely passive flow systems find application in maintaining the long-term evolution of isothermal amplification systems in 0-, 1- and 2-dimensions[6]. Self-assembling rotors made from

magnetic beads may be employed to actively mix solutions in the well-mixed (0-dimensional) case[7]. The scale of practical active valves and mixing modules in microsystems is usually in the range of several hundred micrometers, and there is a problem in integrating the separate control of these elements. There is a need for simple active DNA processing elements with common synchronous control.

Recently, a concept for such an application of steady flow microflow reactors to DNA Computing has been proposed[2]. This differs from the general alternative proposal for using microflow reactors to enhance the scalability of DNA strand routing[8]. It is based on practical experience with large microflow reactor networks, rather than their ideal behavior. A single type of active module, the strand transfer module, was pro-

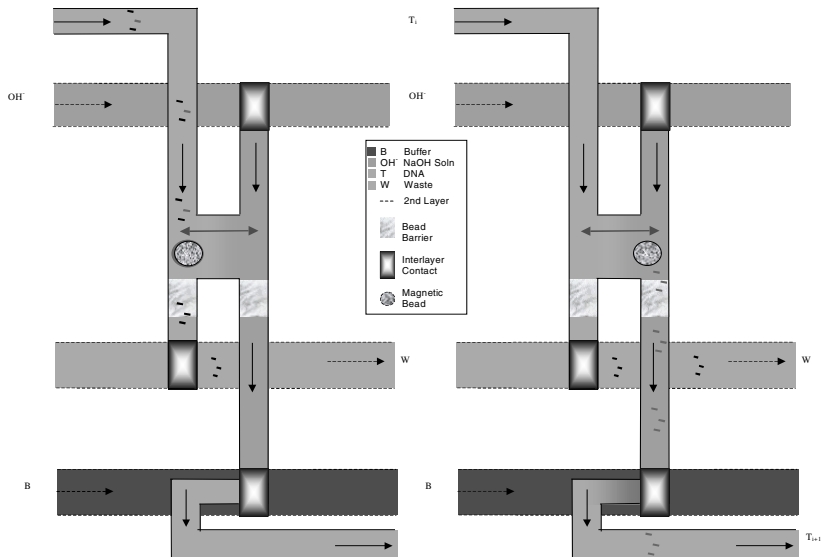


Fig. 1. Alternating states of strand transfer module as proposed in [2]. Selected (red) and non-selected (black) DNA flow in the left solution past beads (only one shown) in a hybridizing buffer. Non binding DNA leaves through the left channel. Upon bead transfer (right hand diagram), bound DNA dissociates from the bead in the non-hybridizing buffer and leaves through the right hand channel before being neutralized (below) for further processing.

posed to implement a hybridization based strand separation step in DNA computing and a scheme for optically programming a network of such modules via photochemistry was introduced. Experimental progress with the development of an optical programming technology for such modules will be reported elsewhere.

The basic function of the module can be seen from Figure 1. Selected DNA strands are to be transferred from one channel to another using two different buffer solutions (e.g. at different pH) and magnetic beads with attached oligonucleotides. Matching DNA templates bind to the beads on the left in continuous flow and are transferred to

the right solution by a magnet which synchronously clocks all module chambers. The release of DNA into the second solution can be achieved by denaturants such as formamide or NaOH. Mixing of the two solutions is limited by the laminar flow. DNA is released in the denaturing solution on the right and leaves via the right hand channel, being neutralized before delivery to a downstream module.

The realization and coupling of such modules presents a number of practical difficulties. In general, connecting many modules with alternating buffer solutions requires channel crossings. The beads must be restrained physically, since a single magnetic control of the entire array is required for good scalability. (The alternative of using inhomogeneous magnetic fields to trap beads at specific locations as employed in is not clearly scalable.) Furthermore, the hydrodynamic flows must be setup to preserve the fluid-fluid boundary in each module. The remainder of this paper is structured as follows. In section 2, we present two microreactor designs and their construction for the strand transfer module.

The second design proved advantageous for the hydrodynamic stability as investigated in section 3, by fluorescent labeling, and should prove useful in improving specificity of strand transfer. The behavior of magnetic beads in the microflow reactors under laminar flow is described in section 4. The paper concludes with a brief discussion of the results and implications for the strand transfer module. Further results with fluorescently labeled DNA binding and strand transfer should be available at the conference.

2. Reactor Design and Construction

The design implementation of the basic module involved the electronic specification of four photomasks (designed via Mentor Graphics Boardstation Software and produced by) which were then transferred to 100 Si wafers (400 μ m thick) via a twin depth etching and double sided structuring of the wafers using TMH and KOH (cf [6]). Channel widths varied between 50 and 200 μ m and depths of 50-100 μ m were employed. At certain locations, the structures were etched right through to make connections for channel crossovers on the reverse side. Planar barriers of only 10 μ m depth were employed to restrain the passage of 15-30 μ m diameter beads and no problem with blocking in filtered buffer solutions were observed. After microstructuring, the Si wafer was anodically bonded to two 500 μ thick pyrex (borosilicate) wafers with thermal expansion coefficients matched to Si. Ultrasonically drilled holes in these pyrex wafers allowed the connection of capillary tubing using UV-hardening glues as described previously, to connect the channels of the microreactor with external fluids.

An initial design of the microreactor, as shown schematically in figure 1, suffered from insufficient hydrodynamic stability (see section 3) and so a further design iteration was performed in which a central channel (washing channel) for a neutral buffer solution was introduced. Because scalability of the reactor module was a key issue, we included some coupled modules already in this initial design phase. The mask overlay design of five single and coupled modules is shown in Figure 2.

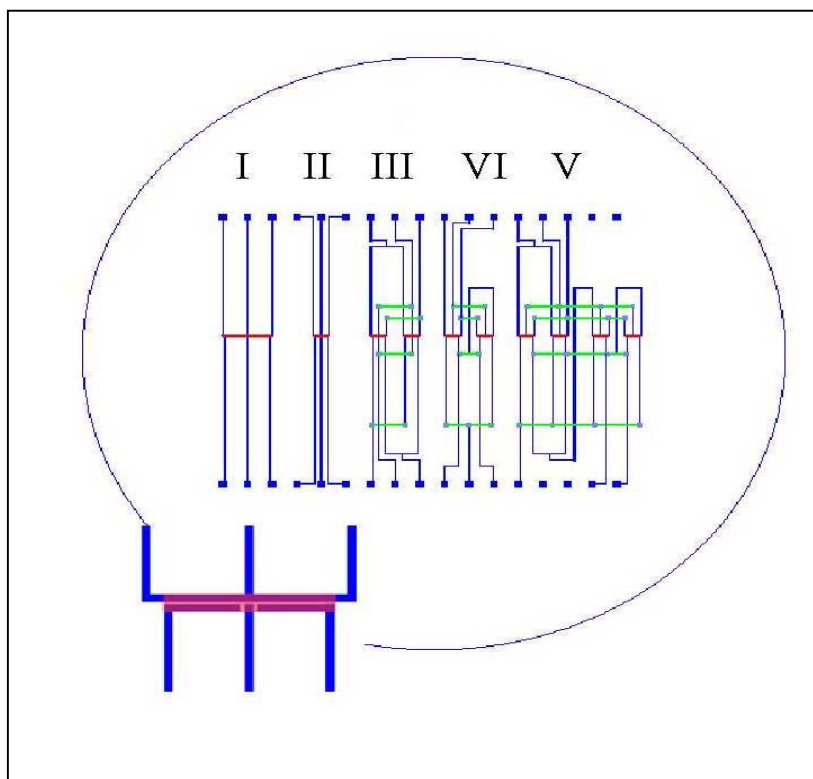


Fig. 2. Mask overlay design for five microflow reactors on a single Si wafer. At the bottom left is an inset showing the detail of the bead barrier. Flow is from top to bottom, and the bead barrier is shown in purple. Microfluidic inputs are in the top row and outputs in the bottom row (3.5mm spacing). Some horizontal channels are on the reverse side of the wafer to avoid contact at channel crossovers.

Reactors I and II differ in the width of the bead transfer module. Reactors III contains two parallel strand transfer modules and reactor IV contains two serial transfer modules. Reactor V has a combination of two parallel and two serial modules.

A photograph of a section of the completed reactor is shown in figure 3. The channel connections on the reverse side of the reactor are not visible. The bead barriers are too fine (25 μ m) and shallow (10 μ m) to be visible. The connection technology is clearly visible with 400 μ m diameter polyethylene capillary tubing. The structures are significantly larger than is necessary for operation, reflecting the early stage of the development.

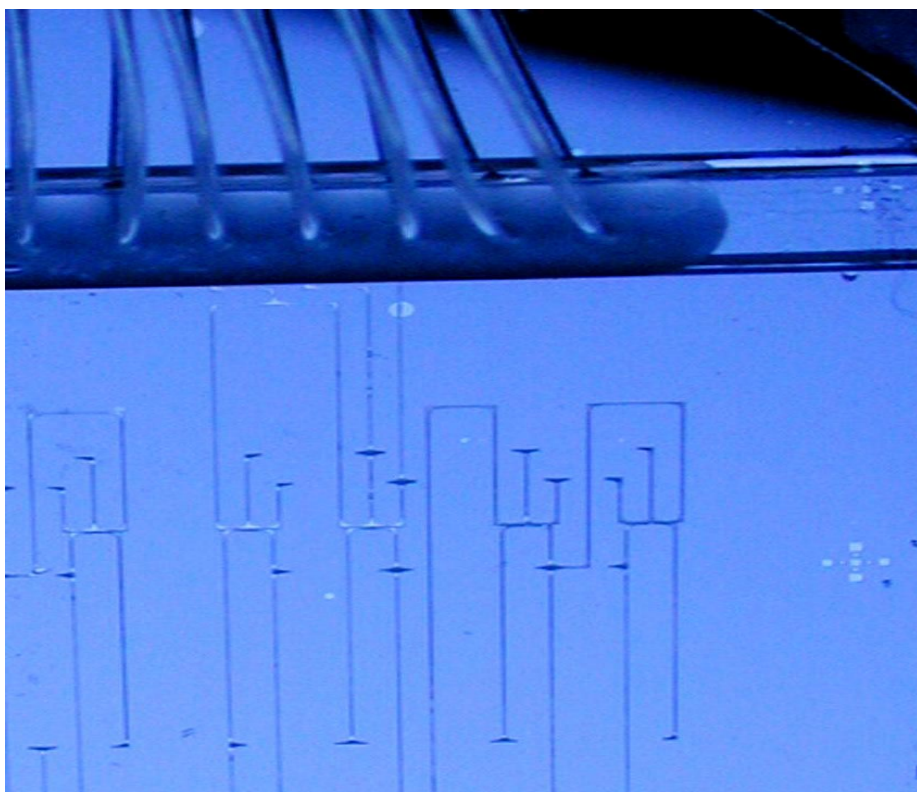


Fig. 3. Image of part of 5 microflow reactor wafer, as specified by the masks of fig. 2.

The connections to capillary tubing is shown above. The spacing between tubes is 3.5mm. A plexiglas stabilizer is glued with a UV-hardening glue to the outer glass wafer after polyethylene capillaries are inserted through drilled holes in the plexiglas (800 μ m) above the holes in the pyrex (300 μ ultrasound drilled).

On the right are markers for mask alignment. The top of the rightmost microreactor V and part of IV are shown. The darker bars above and below the bead barriers (central) involve etching through to channels on the reverse side of the Si wafer (not shown). The back side is also sealed by a pyrex wafer, this time without connections to external tubing.

3. Hydrodynamic Test of Reactor

Fluorescent dyes provide a convenient method of distinguishing between two different solutions in microreactors. In order to test the hydrodynamics stability of the flow in the reactor we have pumped a rhodamine 6G solution by a precision syringe pump at different flow rates via the right channel. In the left picture one can see the diffusion of the rhodamine 6G solution when the pumping speed is zero. In the right hand picture the pumping speed is 4 μ l/min. As one can see on fig. 4, there is a clear separation between the flow into a right channel and the flow into the left channel. A small amount of dye is divided in the central channel (introduced for washing and stability). This stable flow can be achieved by simultaneously pumping both solutions at equal rates (by volume), and does not require an individual regulation. This is

important for the integration to multiple modules. A test of the parallel reactor module (II in fig. 2) showed similar results (data not shown).

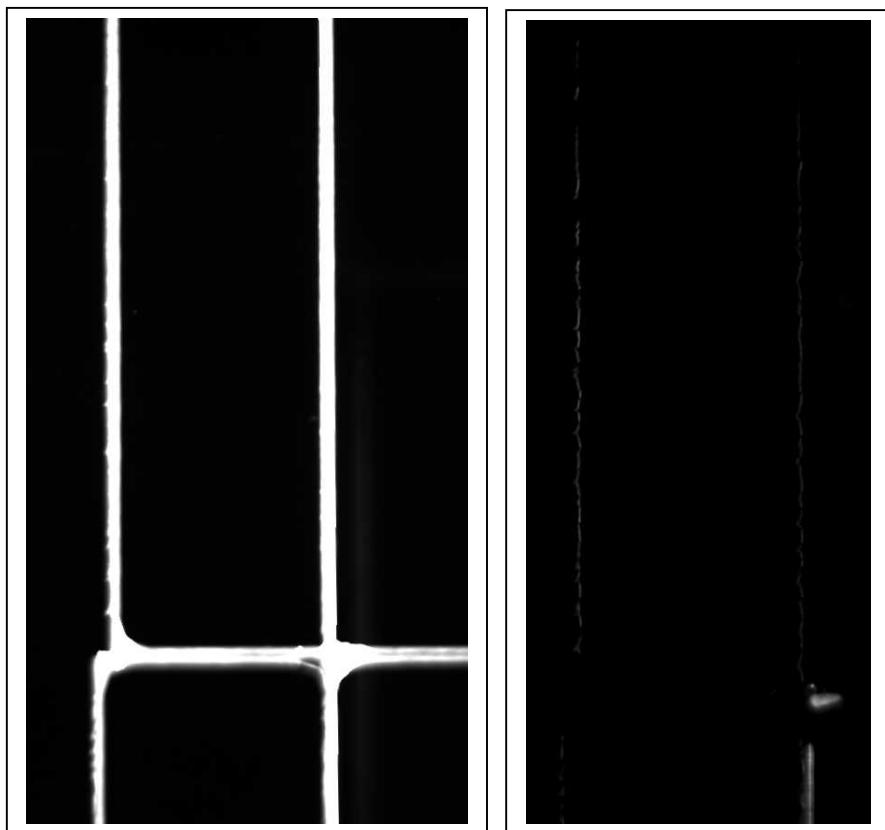


Fig. 4. Fluorescence image of flow in strand transfer module. Flow is from bottom to top and the dye enters on the right channel. Left image: zero flow rate, no separation of solutions. Right image: finite flow rate (see text), no dye in left channel.

4. Active Switching of Magnetic Beads

In addition to the hydrodynamic stability of the different buffer solutions, the strand transfer module has to allow the reliable restraint and switching of magnetic beads between the two solutions. We incorporated latex paramagnetic beads with a diameter of $30\mu\text{m}$ (obtained from Micromod Ltd) into the microreactor. The beads were injected using a syringe pump (as for fluid pumping above), and restrained to the desired microreactor by the bead barrier (a ledge $10\mu\text{m}$ in depth). In contrast with individual module manipulation of beads, the bead restraint is designed to allow the parallel processing of beads in multiple modules in the microreactor. Bead delivery was also successful in the presence of channel crossovers.

The switching behavior of the beads is shown in Fig. 5. A strong magnetic field gradient was employed (SrCo, Maurer Magnetic, Grüningen, Switzerland) was employed. This was particularly important for smaller beads (data not shown). For this experiment, the original microreactor design, with 200 μm channels and no third input for stability was employed. Similar behavior is recorded in the revised microreactor design (see section 2).

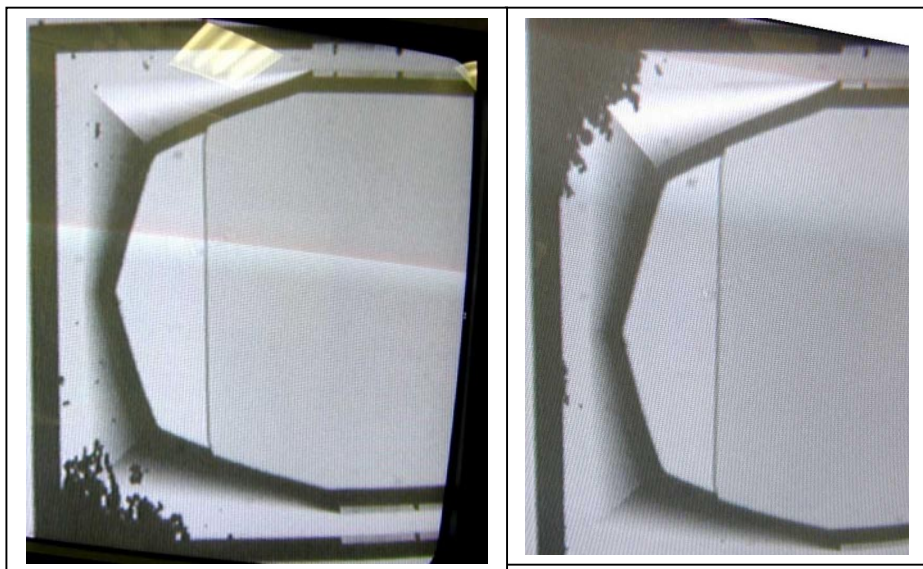


Fig. 5. Switching of magnetic beads between two buffer solutions in microflow module. Flow is from the right and the bead barrier is on the left of each picture. The left hand picture shows the situation with the magnet below, the right hand image the same for the magnet above. Single beads may be discerned. Under-etching of the silicon channel structures with the anisotropic etch method can be seen. This assists the mobility of the beads.

Magnetic beads as small as 8 μm can be used with bead barriers down to 5 μm . Below this size, clogging of the barriers becomes a significant problem.

5. Conclusion

The above results demonstrate the first step in the experimental realization of strand transfer modules for DNA Computing. Attention has been paid to the true scalability of the concept, and the design is based on practical experience with microreactor design. It should be possible to integrate thousands of such modules on a single Si wafer. A concept for DNA Computing based on such a module has been proposed previously. While the authors are aware of concerns over errors in bead based separation of DNA, the microflow reactor framework provides many opportunities for kinetic control which may prove beneficial here. Initial results in this direction have been presented recently⁵. The next stage of experimental results involving DNA bound to beads and selective transfer should be evaluated at the conference.

Acknowledgements. This work would not have been possible without the Herculean effort of B. Streeck and his team at the GMD in constructing the new clean room facilities there. The authors also wish to thank R. Fuchslin for his assistance in selecting suitable magnets.

References

- 1 Liu, Q., Wang, L., Frutos, A.G., Condon, A.E., Corn, R.M. and Smith, L.M.(2000) *Nature* **403** 175-9.
- 2 McCaskill, J.S. (2000) Optically programmable DNA Computing in microflow reactors. *Biosystems* in press.
- 3 Adleman, L.M. (1994) Molecular computation of solutions to combinatorial problems. *Science* **266** 1021-1024.
- 4 Weigl, B.H. and Yager, P. (1999) Microfluidic diffusion-based separation and detection. *Science* **283** 346-7.
- 5 McCaskill J.S. (1997) Spatially resolved in vitro molecular ecology. *Biophys. Chem.* **66** 145-158.
- 6 Schmidt, K., Foerster, P., Bochmann, A. and McCaskill, J.S. (1997) A microflow reactor for two dimensional investigations of in vitro amplification systems. In 1st Int. Conf. Microreaction Tech. (Dechema e.V. Frankfurt).
- 7 Schmidt, K. and McCaskill, J.S. (1998) "Schaltbarer dynamischer Mikromischer mit minimalen Totvolumen" PCT/EP98/03942 .
- 8 Gehani. A. and Reif, J. (1999) "Micro flow bio-molecular computation" *Biosystems* **52** 197-216.
- 9 Fan, Z.H., Mangru, S., Granzow, R., Heaney, P., Ho, W., Dong, Q. and Kumar, R. (1999) Dynamic DNA hybridization on a chip using paramagnetic beads. *Anal. Chem.* **71** 4851-4859.

Author Index

- Ackermann, Jörg 263
Adleman, Leonard M. 27
Arita, Masanori 17
- Bonizzoni, Paola 117
Braich, Ravinderjit S. 27
- Chelyapov, Nickolas 27
Chen, Junghuei 247
Chen, Kevin 199
- De Felice, Clelia 117
Deaton, Russell J. 231
Díaz, Sergio 209
- Eng, Tony 63
Esteban, Juan Luis 209
- Freund, Franziska 130
Freund, Rudolf 130
Frisco, Pierluigi 43
- Gohlke, Marlies 263
Goode, Elizabeth 247
Gouzu, Hidetaka 17
- Hagiya, Masami 17, 89
Hwang, Darryl 27
- Johnson, Cliff 27
- Knight, Jr., Thomas F. 1
Komiya, Ken 17
- LaBean, Thomas H. 145, 173
- Margenstern, Maurice 53
Mauri, Giancarlo 117
McCaskill, John S. 103, 263
- Niemann, Ulrich 103
Nishikawa, Akio 17
- Ogihara, Mitsunori 209
- Penchovsky, Robert 263
- Ramachandran, Vijay 199
Reif, John H. 147, 173
Rogozhin, Yurii 53
Rose, John A. 231
Rothmund, Paul W.K. 27
Rozenberg, Grzegorz 63
Rücker, Thomas 263
- Sakakibara, Yasubumi 220
Sakamoto, Kensaku 17
Seeman, Nadrian C. 173
- Weiss, Ron 1
Winfree, Erik 63
Wood, David Harlan 247
- Yokoyama, Shigeyuki 17
- Zizza, Rosalba 117