

Luca Cardelli  
William Shih (Eds.)

LNCS 6937

# DNA Computing and Molecular Programming

17th International Conference, DNA 17  
Pasadena, CA, USA, September 2011  
Proceedings

 Springer

*Commenced Publication in 1973*

Founding and Former Series Editors:

Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

David Hutchison

*Lancaster University, UK*

Takeo Kanade

*Carnegie Mellon University, Pittsburgh, PA, USA*

Josef Kittler

*University of Surrey, Guildford, UK*

Jon M. Kleinberg

*Cornell University, Ithaca, NY, USA*

Alfred Kobsa

*University of California, Irvine, CA, USA*

Friedemann Mattern

*ETH Zurich, Switzerland*

John C. Mitchell

*Stanford University, CA, USA*

Moni Naor

*Weizmann Institute of Science, Rehovot, Israel*

Oscar Nierstrasz

*University of Bern, Switzerland*

C. Pandu Rangan

*Indian Institute of Technology, Madras, India*

Bernhard Steffen

*TU Dortmund University, Germany*

Madhu Sudan

*Microsoft Research, Cambridge, MA, USA*

Demetri Terzopoulos

*University of California, Los Angeles, CA, USA*

Doug Tygar

*University of California, Berkeley, CA, USA*

Gerhard Weikum

*Max Planck Institute for Informatics, Saarbruecken, Germany*



Luca Cardelli William Shih (Eds.)

# DNA Computing and Molecular Programming

17th International Conference, DNA 17  
Pasadena, CA, USA, September 19-23, 2011  
Proceedings

## Volume Editors

Luca Cardelli  
Microsoft Research  
7JJ Thomson Avenue, Cambridge CB3 0FB, UK  
E-mail: luca@microsoft.com

William Shih  
Dana-Farber Cancer Institute  
Department of Biological Chemistry and Molecular Pharmacology  
44 Binney Street, Boston, MA 02115, USA  
E-mail: william.shih@wyss.harvard.edu

ISSN 0302-9743 e-ISSN 1611-3349  
ISBN 978-3-642-23637-2 e-ISBN 978-3-642-23638-9  
DOI 10.1007/978-3-642-23638-9  
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2011935217

CR Subject Classification (1998): F.1, F.2.2, J.3, E.1, I.2, G.2

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

© Springer-Verlag Berlin Heidelberg 2011

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. Violations are liable to prosecution under the German Copyright Law.

The use of general descriptive names, registered names, trademarks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

*Typesetting:* Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media ([www.springer.com](http://www.springer.com))

# Preface

This volume contains the papers presented at DNA 17: the 17th International Conference on DNA Computing and Molecular Programming held during September 19-23, 2011 in Pasadena, California, USA.

This conference provides a forum for presenting results in the emerging field of biomolecular computation, with an emphasis on DNA as a material for carrying out information processing and material construction on the nanometer scale. Attending scientists have diverse backgrounds but share an interest in practical and theoretical issues relevant for sustaining a trend of exponentially increasing complexity in programmed biomolecular systems.

The conference is held under the auspices of the International Society for Nanoscale Science, Computation, and Engineering (ISNSCE). The DNA 17 Program Committee received 45 paper submissions, of which 12 were selected for oral presentation and inclusion in the proceedings, and an additional 11 for oral presentation; many of the remaining submissions were selected for poster presentations, along with other submissions where a poster-only request was made.

The scientific program included tutorials by Dave Doty (“Theory of Algorithmic Self-Assembly with DNA Tiles”), David Soloveichik (“The Programming Language of Chemical Kinetics, and How to Discipline Your DNA Molecules Using Strand Displacement Cascades”), and Eric Klavins (“Specification and Control of Stochastic Biochemical Systems”), as well as an all-day wet lab tutorial organized by Elisa Franco, Jongmin Kim, and Josh Bishop (“Genelets: Synthetic In Vitro Transcriptional Circuits”).

In addition to contributed talks and poster sessions, five plenary talks were given by Vincent Danos (“Cooperative Assembly Systems”), Yamuna Krishnan (“An Autonomous DNA Nanodevice Captures pH Maps of Living Cells in Culture and In Vivo”), Niles Lehman (“A Highly Cooperative Network of RNA Molecules”), Jack Lutz (“The Computer Science of Molecular Programming”), and Hao Yan (“Designer DNA Architectures for Bionanotechnology”).

Four discussion panels were also held during the conference: Visions for DNA Computing and Molecular Programming (Luca Cardelli, Eric Klavins, Nadrian Seeman, Erik Winfree), Hard Problems in DNA Computing and Molecular Programming (Anne Condon, Vincent Danos, Jack Lutz, John Reif), Applications of DNA Computing and Molecular Programming (Andy Ellington, Yamuna Krishnan, Niles Pierce, Hao Yan), and Interfaces to DNA Computing and Molecular Programming (Deborah Fygenson, Kurt Gothelf, Niles Lehman, Paul Rothemund).

The editors would like to thank the members of the Program Committee and the reviewers for their hard work in reviewing papers and providing comments to the authors. They also thank the members of the Organizing Committee and Steering Committee, and particularly Erik Winfree and Natasha Jonoska,

the respective Committee Chairs, for invaluable advice. The organizers would like to thank the administrative and computer support provided by Lucinda Acosta, Kevin Wong, Briana Ticehurst, Trity Pourbahrani, John Lilley, and Gary Waters. Finally, we would like to thank all the sponsors of the conference, authors, attendees, and supporting staff for making the conference successful and enjoyable.

July 2011

Luca Cardelli  
William Shih

# Organization

DNA 17 was organized by the California Institute of Technology, in cooperation with the International Society for Nanoscale Science, Computation, and Engineering (ISNSCE).

## Steering Committee

Natasha Jonoska (Chair)	University of South Florida, USA
Leonard Adleman	University of Southern California, USA
Luca Cardelli	Microsoft Research Cambridge, UK
Anne Condon	University of British Columbia, Canada
Masami Hagiya	University of Tokyo, Japan
Lila Kari	University of Western Ontario (UWO), Canada
Chengde Mao	Purdue University, USA
Giancarlo Mauri	University of Milan, Italy
Satoshi Murata	Tohoku University, Japan
John Reif	Duke University, USA
Grzegorz Rozenberg	University of Leiden, The Netherlands
Nadrian Seeman	New York University, USA
Andrew Turberfield	Oxford University, UK
Erik Winfree	California Institute of Technology, USA

## Organizing Committee

Erik Winfree (Chair)	California Institute of Technology, USA
Niles Pierce	California Institute of Technology, USA
Damien Woods	California Institute of Technology, USA
David Doty	California Institute of Technology, USA

## Program Committee

Luca Cardelli (Co-chair)	Microsoft Research Cambridge, UK
William Shih (Co-chair)	Harvard Medical School, Dana-Farber Cancer Institute, Wyss Institute, USA
Anne Condon	University of British Columbia, Canada
David Doty	University of Western Ontario, Canada
Shawn Douglas	Harvard University, USA
Andrew Ellington	The University of Texas at Austin, USA
Max Garzon	The University of Memphis, USA
Masami Hagiya	University of Tokyo, Japan
Natasha Jonoska	University of South Florida, USA
Ming-Yang Kao	Northwestern University, USA



Lila Kari	University of Western Ontario, Canada
Eric Klavins	University of Washington, USA
Satoshi Kobayashi	University of Electro-Communications, Japan
Dongsheng Liu	Tsinghua University, China
Chengde Mao	Purdue University, USA
Satoshi Murata	Tohoku University, Japan
Jacques Nicolas	INRIA/IRISA, France
Pekka Orponen	Aalto University, Finland
John Reif	Duke University, USA
Yannick Rondelez	The University of Tokyo, Japan
Yasubumi Sakakibara	Keio University, Japan
Georg Seelig	University of Washington, USA
Friedrich Simmel	Technical University Munich, Germany
David Soloveichik	California Institute of Technology, USA
Darko Stefanovic	University of New Mexico, USA
Fumiaki Tanaka	University of Tokyo, Japan
Andrew Turberfield	University of Oxford, UK
Erik Winfree	California Institute of Technology, USA
Damien Woods	California Institute of Technology, USA
Hao Yan	Arizona State University, USA
Bernard Yurke	Boise State University, USA
Byoung-Tak Zhang	Seoul National University, Korea
David Zhang	Harvard University, USA

## Referees

Bishop, Josh	Chandran, Harish	Chen, Yi
Conway, Nicholas	Cui, Bo	Fanning, Leigh
Garg, Sudhanshu	Gopalkrishnan, Nikhil	Gu, Hongzhou
Hamada, Shogo	Kawamata, Ibuki	Kim, Byoung-Hee
Kobayashi, Satoshi	Masson, Benoit	Oishi, Kevin
Olah, Mark	Sadat, Sayem	Schaeffer, Joseph
Schaus, Thomas	Schmidt, Thorsten	Semenov, Oleg
Simjour, Amir	Zhang, Chuan	

## Sponsoring Institutions

The Donna and Benjamin M. Rosen Center for Bioengineering at Caltech  
 The United States National Science Foundation (NSF) Computer and Information Science and Engineering (CISE) directorate  
 The California Institute of Technology

# Table of Contents

## Invited Talks

Cooperative Assembly Systems . . . . .	1
<i>Vincent Danos, Heinz Koepl, and John Wilson-Kanamori</i>	
The Computer Science of Molecular Programming . . . . .	21
<i>Jack H. Lutz</i>	
An Autonomous DNA Nanodevice Captures pH Maps of Living Cells in Culture and <i>in Vivo</i> . . . . .	22
<i>Sunaina Surana, Souvik Modi, and Yamuna Krishnan</i>	
Cooperation in an All-RNA Network . . . . .	32
<i>Nilesh Vaidya, Jessica Mellor, and Niles Lehman</i>	
Designer DNA Architectures for Bionanotechnology . . . . .	33
<i>Hao Yan</i>	

## Contributed Papers

An Improved DNA-Sticker Addition Algorithm and Its Application to Logarithmic Arithmetic . . . . .	34
<i>Mark G. Arnold</i>	
Graph-Theoretic Formalization of Hybridization in DNA Sticker Complexes . . . . .	49
<i>Robert Brijder, Joris J.M. Gillis, and Jan Van den Bussche</i>	
Localized Hybridization Circuits . . . . .	64
<i>Harish Chandran, Nikhil Gopalkrishnan, Andrew Phillips, and John Reif</i>	
Less Haste, Less Waste: On Recycling and Its Limits in Strand Displacement Systems . . . . .	84
<i>Anne Condon, Alan Hu, Ján Mañuch, and Chris Thachuk</i>	
One-Dimensional Staged Self-assembly . . . . .	100
<i>Erik D. Demaine, Sarah Eisenstat, Mashhood Ishaque, and Andrew Winslow</i>	
Computing Maximal Kleene Closures That Are Embeddable in a Given Constrained DNA Language . . . . .	115
<i>Stavros Konstantinidis and Nicolae Santean</i>	

Modelling, Simulating and Verifying Turing-Powerful Strand Displacement Systems . . . . .	130
<i>Matthew R. Lakin and Andrew Phillips</i>	
Synthesizing Small and Reliable Tile Sets for Patterned DNA Self-assembly . . . . .	145
<i>Tuomo Lempiäinen, Eugen Czeizler, and Pekka Orponen</i>	
Multivalent Random Walkers — A Model for Deoxyribozyme Walkers . . . . .	160
<i>Mark J. Olah and Darko Stefanovic</i>	
Exact Shapes and Turing Universality at Temperature 1 with a Single Negative Glue . . . . .	175
<i>Matthew J. Patitz, Robert T. Schweller, and Scott M. Summers</i>	
Autonomous Resolution Based on DNA Strand Displacement . . . . .	190
<i>Alfonso Rodríguez-Patón, Iñaki Sainz de Murieta, and Petr Sosík</i>	
Multiple Molecular Spiders with a Single Localized Source— The One-Dimensional Case (Extended Abstract) . . . . .	204
<i>Oleg Semenov, Mark J. Olah, and Darko Stefanovic</i>	
<b>Author Index</b> . . . . .	217

# Cooperative Assembly Systems\*

Vincent Danos<sup>1,\*\*</sup>, Heinz Koepl<sup>2</sup>, and John Wilson-Kanamori<sup>1</sup>

<sup>1</sup> LFCS, School of Informatics, University of Edinburgh  
vdanos@inf.ed.ac.uk

<sup>2</sup> ETH Zurich

**Abstract.** Several molecular systems form large-scale objects. One would like to understand their assembly and how this assembly is regulated. As a first step, we investigate the phase transition structure of a class of bipartite cooperative assembly systems. We characterize which of these systems have a (probabilistic) equilibrium and find an explicit form for their local energy ( 2). We obtain, under additional limitations on cooperativity, the average dynamics of some partial observables ( 4). Combining both steps, we obtain conditions for the phase transition to a large cluster ( 5).

Numerous intracellular molecular systems (post-synaptic densities, bacterial chemotaxis sensors, or focal adhesion complexes) form extensive structures which are somewhat intermediate between traditional signalling complexes and large-scale objects more familiar from statistical physics. One would like to understand the specific regulation controlling their assembly and how this regulation relates to their processing information.

As a first step, this paper investigates a class of bipartite cooperative assembly systems and examines whether one can obtain conditions for *criticality*, defined here as appearance of a large cluster. We characterize which of these systems have a probabilistic equilibrium and find an explicit form for their associated (local) energy functional. Then, drawing on recent ideas from rule-based model reduction [7], we obtain under additional conditions limiting the type of cooperativity further, a differential system describing the average dynamics of some partial observables appearing in the energy functional. This gives us in particular their average steady state concentrations (as implicit functions of the cooperativity parameters). By combining both steps, we can derive a criticality condition for the appearance of a large cluster. The said condition uses the mean reproduction rate of a suitable branching process (this is standard in the statistical physics literature [5]).

This improves on earlier work considering the non-cooperative case [10]. Section 3 incorporates numerical experiments - using a recent implementation KaSim of the Kappa rule-based modeling language. The example model is described in the Appendix. Simulation files and further useful information can be found here.

---

\* This paper is an invited contribution to the Proceedings of the 17th International Conference on DNA Computing and Molecular Programming (2011).

\*\* Corresponding author.

# 1 The Cooperative Assembly Model

The assembly model we study considers only two types of agents or nodes,  $A$  and  $B$  - although some of the conclusions we attain are clearly generalizable. We suppose each agent is equipped with a set of identical sites, namely, agents of type  $A$  have  $v_A$  sites of type  $a$ , and agents of type  $B$  have  $v_B$  sites of type  $b$ . This is only to make the model tractable and carries no deep meaning.

Sites of type  $a$  and  $b$  can bind according to rules which we describe below, and we call the resulting objects *site graphs*, to insist that linking is done via the use of sites and not directly on nodes, as with usual graphs. One can think of sites as domains of proteins, or DNA domains, or other media as used in DNA-based programmable chemistry, as long as elements glue along well-defined interfaces (see eg [13]) and steric problems are not a concern. Saying that sites are identical means that the rules will not distinguish various occurrences of  $a$  or  $b$ . We refer to  $v_A, v_B$  as the *valences* of  $A, B$ , and assume  $v_A, v_B > 0$ . We write  $n_A$  for the total number of agents of type  $A$ , and  $n_B$  for that of type  $B$ , assume  $n_A, n_B > 0$ ; we write  $\mathcal{G}(n_A, n_B)$  for the set of bipartite site graphs with  $n_A, n_B$  nodes of type  $A, B$ , and  $x_0$  for the unique site graph in  $\mathcal{G}(n_A, n_B)$  which has no links. This site graph will serve as our origin and we will assign energy zero to it.

The rules are as follows. For each pair  $0 \leq i < v_A, 0 \leq j < v_B$ , we assume a reversible association rule,  $r(i, j)$ , whereby one can bind a pair of nodes of types  $A, B$ , and respective degrees  $i$  and  $j$ , via a pair of free sites  $a, b$ ; or unbind an already existing bond. We write  $\gamma_{i,j}^+, \gamma_{i,j}^- > 0$  respectively for the association and dissociation rate of  $r(i, j)$ . These are the rates at which the corresponding rule is applied to one instance. This defines a continuous-time Markov chain in the usual fashion. We set  $\Gamma_{i,j} = \gamma_{i,j}^- / \gamma_{i,j}^+ \in \mathbb{R}_+$ , and write  $\Gamma$  for the associated  $v_A \times v_B$  matrix.

As sites of same type cannot bind, every site graph reachable from  $x_0$  is bipartite (Fig. 3), and so is in  $\mathcal{G}(n_A, n_B)$ . Conversely, every site graph in  $\mathcal{G}(n_A, n_B)$  can be reached from the totally disconnected state  $x_0$  (hence from any other one, by symmetry of the transitions) by creating all the needed connexions. Hence, the continuous-time Markov chain underpinning the dynamics is irreducible and positive-recurrent, and, its state space  $\mathcal{G}(n_A, n_B)$  being finite, it has a unique invariant probability.

Note that, as soon as  $v_A, v_B \geq 2$ , the dynamics can build polymers, meaning connected site graphs the size of which is only bounded by the total number of available nodes  $N = n_A + n_B$  (Fig. 3).

Depending on the definition of an instance of a rule, the set of graphs reachable from  $x_0$  might include double  $AB$  bonds, which we call ‘alkenes’ in this paper. The model, presented in the appendix, allows double bonds because the Kappa rules are shorter to write, but as this complicates things a bit on the theory side, we suppose here that rules can only bind sites from nodes that are not yet sharing an edge. We will see in 3 that the discrepancy in simulations is negligible (Fig. 4).

## 2 Equilibrium

Throughout the paper we write  $\log(x)$  for the *natural* logarithm,  $\binom{n}{m}$  for the binomial coefficient,  $[m; n] = m! \binom{n}{m}$  for the number of injections of  $m$  into  $n$ , where  $m, n \geq 0$ , and  $\binom{n}{n_1 \dots n_k}$  with  $n_i \geq 0$ ,  $n_1 + \dots + n_k = n$  for the multinomial coefficient.

Recall Stirling's formula  $\log n! \sim n \log n - n$ , from which follows that:

$$\log \binom{n}{n_1 \dots n_k} \sim nH(n_i/n)$$

where  $H(p) = -\sum_i p_i \log p_i$  is the Shannon entropy of a probability  $p$  with finite support. We will use this later in this section.

Our first step is to characterize the models for which the invariant probability is an equilibrium (aka has detailed balance), as when it is the case, this will give us a computational handle on the behaviour of the system.

**Proposition 1.**  *$\Gamma$  has an equilibrium iff  $\Gamma = \Gamma_A \Gamma_B^t$  for some  $\Gamma_A \in \mathbb{R}_+^{v(A)}$ ,  $\Gamma_B \in \mathbb{R}_+^{v(B)}$ .*

*Proof.* Consider two states  $x, y$  in  $\mathcal{G}(n_A, n_B)$  such that there exists a (one-step) transition from  $x$  to  $y$  (and therefore from  $y$  to  $x$ , by reversibility of rules).

Without loss of generality, suppose  $y$  is obtained from  $x$  by adding a bond between nodes  $u, v$  of respective types  $A$  and  $B$ .

Both  $u, v$  have a definite degree in  $x$ , say  $i$  and  $j$ , and there is therefore only one rule the application of which underlies this transition, namely  $r(i, j)$ . The ratio of the backward and forward rates is given by:

$$\rho(x, y) = \frac{\Gamma_{i,j}}{(v_A - i)(v_B - j)} \quad (1)$$

Indeed, in order to bind nodes  $u, v$  of  $x$ , one has to choose one of the  $v_A - i$  free sites of  $u$ , and the same for  $v$ . As  $i < v_A$ , and  $j < v_B$  the above formula is always defined.

By connectedness of the transition graph, over  $\mathcal{G}(n_A, n_B)$ , we know that there is at most one energy function  $V$  which describes the equilibrium (it might not exist but it is unique), up to an additive constant.

Setting  $V(x_0) = 0$  (which one can always do as energy is defined to an additive constant), we know that if  $V$  exists, it must verify:

$$V(x) = \sum_{(z, z') \in \phi: x_0 \rightarrow x} \log \rho(z, z') \quad (2)$$

where  $\phi$  is a sequence of successive transitions leading from  $x_0$  to  $x$ . For  $V$  to be consistent, we need this definition to be independent of the choice of  $\phi$ . It is enough to show that two successive bindings on the same node  $u$  obtain the same  $\Delta V$ , as this is the only case where two instances of rules interact in a non trivial way (up to reversibility).

So, suppose  $u : A$  has degree  $i$  in  $x$  and gets bound successively to  $v_1 : B$  of degree  $j_1$ , to obtain the intermediate  $y$ , and then to  $v_2 : B$  of degree  $j_2$ , to obtain  $z$ . (As we do not allow alkene formation,  $v_1 \neq v_2$ .)

The product of the rate ratios  $\rho(x, y_1)\rho(y_1, z)$  along the two transitions is:

$$\frac{\Gamma_{i,j_1}\Gamma_{i+1,j_2}}{(v_A - i)(v_B - j_1)(v_A - i - 1)(v_B - j_2)}$$

Path-independence forces the above to be constant under exchanging the roles of  $v_1$  and  $v_2$ , which leads to the following simple constraint (note that the denominator does not depend on the order):

$$\Gamma_{i,j_1}\Gamma_{i+1,j_2} = \Gamma_{i,j_2}\Gamma_{i+1,j_1}$$

and by symmetry (exchanging the roles of  $A$  and  $B$ ):

$$\Gamma_{i_1,j}\Gamma_{i_2,j+1} = \Gamma_{i_1,j+1}\Gamma_{i_2,j}$$

It is easy to see that the above two equations hold iff:

$$\Gamma_{0,0}\Gamma_{i,j} = \Gamma_{i,0}\Gamma_{0,j}$$

and the statement follows, eg with  $\Gamma_A(i) = \Gamma_{i,0}\Gamma_{0,0}^{-\frac{1}{2}}$ ,  $\Gamma_B(j) = \Gamma_{0,j}\Gamma_{0,0}^{-\frac{1}{2}}$ .  $\square$

There are a few remarks worth making here: first, allowing for alkenes would not add any further constraints on the rates; second, the equilibrium condition is reminiscent of the condition derived on an earlier paper for the validity of a static analysis of reachability based on views [2] - it forces a simple multiplicative form of the dependency of  $\Gamma$ s in the degrees  $i, j$ ; third, the condition above does not constrain rate constants but only their ratios  $\Gamma_{i,j}$ , thus, it is always possible to multiply jointly  $\gamma_{i,j}^+$  and  $\gamma_{i,j}^-$  by an arbitrary positive scalar, without altering the equilibrium or its existence.

We suppose from now on that  $\Gamma$  satisfies the equilibrium condition.

The proof above, in the case the equilibrium condition is fulfilled, gives an explicit expression for the energy:

**Proposition 2.** *Supposing  $\Gamma$  has an equilibrium, the underlying energy is:*

$$V(x) = \sum_{u \in x} \log \frac{\prod_{0 \leq i < d(u)} \Gamma_{\tau(u)}(i)}{[d(u); v_{\tau(u)}]} \quad (3)$$

with  $\tau(u) \in \{A, B\}$  the type of node  $u$  in  $x$ ,  $v_{\tau(u)}$  its valence, and  $d(u)$  its degree.

*Proof.* We prove the formula by induction on the number of bonds in  $x$ .

If there are none, then  $x = x_0$  and  $V(x_0) = 0$ .

Suppose now this is true for  $x$ , and add a bond in  $x$  between nodes  $u : A$  and  $v : B$  with respective degrees  $i, j$ . The difference of energy incurred by this addition is:

$$V(y) - V(x) = \log \frac{\Gamma_{\tau(u)}(d(u))}{(v_{\tau(u)} - d(u))} + \log \frac{\Gamma_{\tau(v)}(d(v))}{(v_{\tau(v)} - d(v))}$$

which is indeed equal to  $\log \rho(x, y)$  as in Eq.1.  $\square$

As said earlier, the dynamics described by  $\Gamma$  will drive the system to the invariant probability with support  $\mathcal{G}(n_A, n_B)$  associated to the energy  $V(x)$ :

$$p_V(x) = \frac{e^{-V(x)}}{\sum_{y \in \mathcal{G}(n_A, n_B)} e^{-V(y)}}$$

Note that the above calculations hold more generally for any set of homogeneous agents as long as their association/dissociation rules are limited to depend only on the degree of the bindees.

## 2.1 Energy, Entropy

In this subsection, we work out some constraints on the dynamics implied by the specific format of the potential  $V$ , which we have constructed above.

We write  $[A_i; x]$  for the number of *embeddings* into  $x$  of a single agent  $A$  with degree  $i$ ,  $A_i(x) := [A_i; x]/i!$  for the number of *occurrences* of  $A_i$  in  $x$ , and  $\mathbf{A}(x)$ ,  $\mathbf{B}(x)$  for the associated *degree vector*; that is to say  $\mathbf{A}(x) \in \mathbb{R}^{v(A)}$  has  $i$ th component  $A_i(x)$ , and similarly for  $\mathbf{B}(x) \in \mathbb{R}^{v(B)}$ .

It is easy to see that we can rewrite Eq.3 for  $V$  as:

$$V(x) = \sum_{0 < i \leq v_A} \epsilon_A(i) A_i(x) + \sum_{0 < i \leq v_B} \epsilon_B(i) B_i(x) \quad (4)$$

with  $\epsilon_\tau \in \mathbb{R}^{v(\tau)}$  defined component-wise as:

$$\epsilon_\tau(i) := \sum_{0 \leq j < i} \log \frac{\Gamma_\tau(j)}{v_\tau - j}$$

We can write succinctly  $V(x) = \langle \epsilon_A, \mathbf{A}(x) \rangle + \langle \epsilon_B, \mathbf{B}(x) \rangle$ . As a consequence,  $x$ s with the same  $\mathbf{A}(x)$  and  $\mathbf{B}(x)$  are equally likely at equilibrium. In particular swapping links, or permuting nodes of the same type, leaves the energy unchanged. The set of such graphs form an equivalence class. The distinguished state  $x_0$  belongs to a singleton class (for generic values of  $\epsilon_\tau(i)$  that is) which is labelled by the degree vectors  $\mathbf{0}, \mathbf{0}$ .

As  $\sum i A_i(x) = \sum j B_j(x)$  is the total number of bound  $a$  or  $b$  sites in  $x$ , the log of the cardinality of the class of  $x$ , or the class *entropy*, is:

$$S(x) = \log(\sum i A_i(x))! + \log \binom{n_A}{A(x)} + \log \binom{n_B}{B(x)} \quad (5)$$

This way to look at energy, invites one to think of  $\epsilon_\tau(i)$  as setting the penalty/reward for a node of type  $\tau$  to acquire degree  $i$ . We mention in passing that the idea of using a flexible syntax of *local* energy functionals in modeling biomolecular interactions has a long tradition, which was recently revived in Refs. [14, 12]. See Ref. [3] for a similar investigation in the case of ordinary chemical reactions (aka Petri nets).



## 2.2 Cooperativity

The first question we can ask is that of *cooperativity*. The assembly rules manifest cooperativity if adding a bond to an already busy agent results in a smaller change of potential  $\Delta V$  than would adding it to an agent less busy [16]. Adding a bond on a pair  $A_i, B_j$ ,  $i < v_A$ ,  $j < v_B$ , results in a difference of potential:

$$\begin{aligned}\Delta V &= \epsilon_A(i+1) - \epsilon_A(i) + \epsilon_B(j+1) - \epsilon_B(i) \\ &= \log \Gamma_A(i)\Gamma_B(j)/(v_A - i)(v_B - j)\end{aligned}$$

as follows from the definition above.

Hence it is energetically more advantageous to add an  $(i, j)$  bond than an  $(i', j')$  iff:

$$\frac{\Gamma(i, j)}{\Gamma(i', j')} = \frac{\Gamma_A(i)}{\Gamma_A(i')} \frac{\Gamma_B(j)}{\Gamma_B(j')} \leq \frac{v_A - i}{v_A - i'} \frac{v_B - j}{v_B - j'}$$

The conclusion is that it is *not* enough, in order to have cooperativity, to suppose that  $\Gamma_\tau(i)$  is a decreasing function of the degree  $i$  (that would be  $\Gamma_\tau(i)/\Gamma_\tau(i') \leq 1$  when  $i > i'$ ); instead we need to assume that  $\Gamma_\tau(i)$  decreases faster than  $(v_\tau - i)/(v_\tau - i')$  (which is already  $\leq 1$ , again supposing  $i > i'$ ). In particular, a system of rules where  $\Gamma$ s are constant will be anti-cooperative, if slightly. However, as  $\partial v((v - i)/(v - i')) = (i - i')/(v - i')^2 \geq 0$ , when  $i > i'$ , the larger  $v$ , the weaker the effect.

## 2.3 Bond Displacement

Here is another question we can investigate. Suppose  $x$  is a site graph,  $u_1, u_2$  are nodes in  $x$  of the same type  $\tau$ , and respective degrees  $i_1 > 0$ ,  $i_2 < v_\tau$ . We can define a new site graph  $y = (u_1, u_2) \cdot x$  by giving one link of  $u_1$  to  $u_2$ .

**Lemma 1.** *Suppose  $x$  is a site graph and  $y = (u_1, u_2) \cdot x$  is obtained by bond displacement, then, the induced difference of potential  $\Delta V := V(y) - V(x)$  and entropy  $\Delta S := S(y) - S(x)$  are:*

$$\Delta V = \log \frac{\Gamma_\tau(i_2)}{\Gamma_\tau(i_1 - 1)} + \log \frac{v_\tau - i_1 + 1}{v_\tau - i_2}$$

$$\begin{aligned}\Delta S &= \log n_{i_1} n_{i_2} - \log (n_{i_1 - 1} + 1)(n_{i_2 + 1} + 1) \quad \text{if } i_2 + 1 \neq i_1 - 1 \\ &= \log n_{i_1} n_{i_2} - \log (n_k + 2)(n_k + 1) \quad \text{if } k = i_2 + 1 = i_1 - 1\end{aligned}$$

*Proof.* The bond displacement preserves the total number of links, as well as the degrees of nodes not of type  $\tau$ . So, in Eq.5, only the  $\tau$ -side multinomial in  $S$  is changed. The expression for  $\Delta V$  is an easy computation.  $\square$

From the expression for  $\Delta S$ , we see that if the target degrees  $i_1 - 1$ ,  $i_2 + 1$  have lower counts in  $x$  than the source degrees  $i_1$  and  $i_2$ , then  $\Delta S \geq 0$ , meaning the class cardinality increases. Purely based on this entropic contribution, bigger classes, which get a higher probability, will tend to have uniform degree distribution.

What about the energy contribution? Supposing  $\Gamma_\tau(i) = \Gamma$  is constant, we see from the expression for  $\Delta V$  above, that:

$$\begin{aligned}\Delta V &= \log(v_\tau - i_1 + 1) - \log(v_\tau - i_2) \\ \Delta V \leq 0 &\text{ iff } i_2 < i_1\end{aligned}$$

Hence states where degree differences between nodes of type  $\tau$  are  $> 1$  will be penalized. Purely based on this energy contribution (and  $\Gamma_\tau(i)$  constant), graphs where the degree vector is concentrated will be favoured at equilibrium.<sup>1</sup>

The question now is which of the opposite forces wins asymptotically in  $N = n_A + n_B$  the total number of nodes.<sup>2</sup>

## 2.4 A Rough Estimate of $\mathbf{B}(x)$ given $\mathbf{A}(x)$

Suppose, to lighten up the notation, that the  $\Gamma$ -constant type is  $B$ . Suppose further that the  $\mathbf{A}(x)$  distribution is *fixed*, which also fixes the total number of links as  $\sum_i iB_i(x) = \sum_j jA_j(x)$ . We will discuss this latter assumption, which is key, later in 3.

Set  $p_i := B_i(x)/n_B$  for the probability that a randomly chosen  $B$  in  $x$  has degree  $i$ . Write  $q_i := \sum_{i \leq k \leq v_B} p_k$ , the probability that such a  $B$  has degree  $\geq i$  (a non-increasing sequence).

The asymptotic entropic contribution, given  $\mathbf{A}(x)$ , is, up to an additive constant:

$$S(\mathbf{B}(x)) = \binom{n_B}{\mathbf{B}} \sim n_B H(p) \leq n_B \log v_B$$

On the other hand the energy contribution, again given  $\mathbf{A}(x)$ , and using the  $\Gamma_B$  constant assumption, is, up to an additive constant:

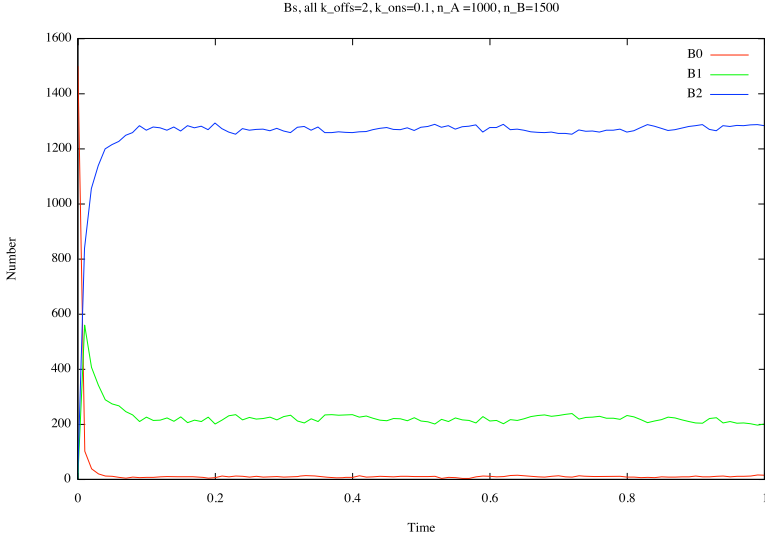
$$\begin{aligned}-V(\mathbf{B}(x)) &= \sum_{0 \leq i \leq v_B} B_i(x) \log[i; v_B] \\ &= n_B \sum_{0 \leq i \leq v_B} p_i \log[i; v_B] \\ &= n_B \sum_{1 \leq k \leq v_B} q_k \log(v_B - k + 1) \\ &= q_1 n_B \log v_B + q_2 n_B \log(v_B - 1) + \dots\end{aligned}$$

The likeliest  $\mathbf{B}(x)$  will minimize the *free energy*  $V(\mathbf{B}(x)) - S(\mathbf{B}(x))$ , given the total number of links. One sees that, asymptotically, and assuming the  $q_1 \sim 1$ , which is definitely going to be the case near criticality, the entropy contribution is trumped by the energy one. See Figs. 1, 2 for concrete examples.

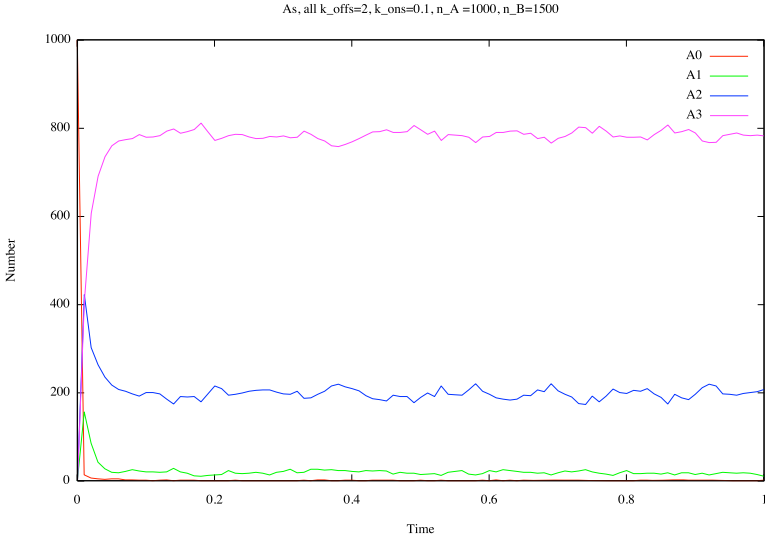
So, in this constant case, it seems reasonable (and all the more reasonable when  $v_B$  is high) to *neglect* the entropic contribution, and approximate the equilibrium distribution for the degree of nodes of type  $B$  by the following *two-valued distribution* which minimizes the potential  $V$  given a fixed number of links  $\sum iB_i$ :

<sup>1</sup> As  $\partial_{v_\tau} \exp(-\Delta V) = (i_1 - 1 - i_2)/(v - i_1 + 1)^2$  has the same sign as  $\Delta V$ , higher valences will increase the degree distribution concentration of nodes of type  $\tau$ .

<sup>2</sup> As an aside, we can observe that the  $\Delta V$  of bond displacement proves that the distribution of degrees on  $\tau$ s with constant  $\Gamma_\tau$  is not a per site independent distribution - given that all other sites of a node  $u$  are free, it will be far likelier that a given site of  $u$  is bound.



**Fig. 1.** We observe  $\mathbf{B}(x)$  along a simulation with parameters:  $\gamma_{i,j}^- = 2$ ,  $\gamma_{i,j}^+ = 0.1$ ,  $n_A = 1000$ ,  $n_B = 1500$ ,  $v_A = 3$ ,  $v_B = 2$ . Note that  $B_0$  rapidly becomes negligible.



**Fig. 2.** We observe now  $\mathbf{A}(x)$  along a simulation with the same parameters as above in Fig. 1. Again  $A_0$  rapidly becomes negligible, and we see that the distribution concentrates on degrees 2 and 3, as in the estimate of 2.4.

- $p_q = 1 - r/n_B$  and  $p_{q+1} = r/n_B$
- where  $q, r \in \mathbb{N}$  are unique such that  $\sum iB_i = qn_B + r, 0 \leq r < n_B$ .

Thus, the  $V$ -minimal assignment obtained by Euclidean division of the total number of bonds by the total number of nodes, will serve as an estimate of the true one.

Encouragingly, simulations in Figs. 1, 2, where both  $\Gamma_A, \Gamma_B$  are constant seem to agree with our estimate. On the  $A$ -side, we see that  $A$ s concentrates on degrees 2 and 3, and the sampled value  $\mathbf{A} = (0, 10, 210, 780)$  compares well with the estimate  $\hat{\mathbf{A}} = (0, 0, 230, 770)$ , which we can use since  $\Gamma_A$  is constant. On the  $B$ -side,  $B$ s concentrate on degrees 1 and 2, and we get  $\mathbf{B} = (15, 201, 1284)$  which again compares well with  $\hat{\mathbf{B}} = (0, 231, 1269)$ , if slightly less so, which is expected as the valence of  $B$  is lower. Nevertheless, as the reader can tell, the estimate is not at all rigorous, and has to be seen with caution. Caution notwithstanding, we will use this in 5.

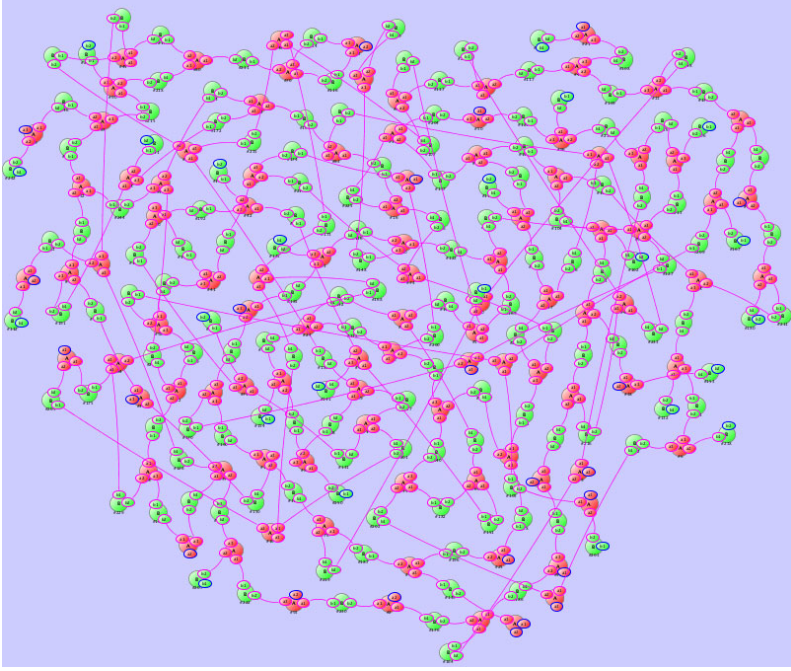
### 3 Simulations

We now look more closely at simulations of some of our assembly systems. As in 2, we choose  $v_A = 3, v_B = 2$  (enough to assemble polymers of unbounded size given unboundedly many nodes). Perfect matchings are reachable only if the stoichiometry is  $2n_A = 3n_B$ ; this is the stickiest the system can get, all other things being equal, as in this case all sites are used to form a bond. Eg if we pick  $n_A, n_B = 100, 150$ , we get an equilibrium probability on  $\mathcal{G}(100, 150)$  of which a typical sample looks as in Fig. 3, indeed almost a perfect matching.

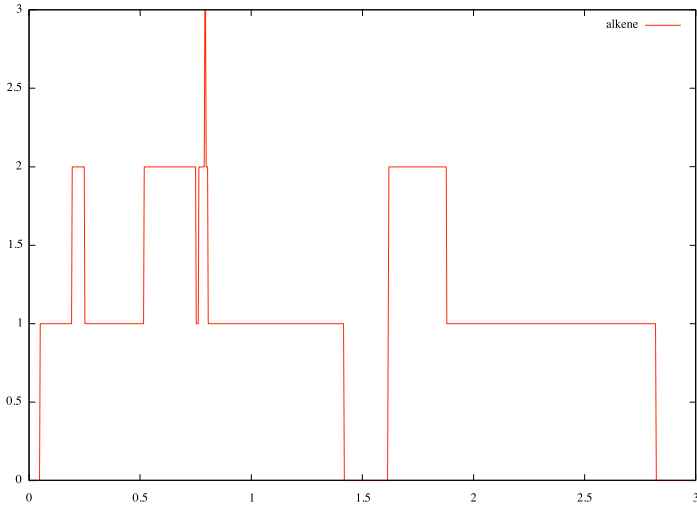
We pick parametrizations where  $\gamma^+$  is constant and  $\gamma^-$  only depends on  $A$ 's degree. Hence,  $\Gamma(i, j) = \Gamma(i)$  does not depend on  $j$  and the conditions for equilibrium given in 2 are clearly satisfied. As  $\Gamma_B$  is constant, the additional assumption introduced in 2.4 also is. We choose  $\Gamma_A(i)$  as a monotonic function of  $i$  to simulate both cooperativity and anti-cooperativity (with the provision discussed in 2.2). We see in the resulting simulations in Fig. 5 that alkenes are extremely rare, and that, unsurprisingly, the cooperative case is 'stickier' (has a higher edge count).

Another thing one sees, is that, clearly, there is no good way to talk about a deterministic steady state of such systems, as for large populations, the state space accessible to probabilistic equilibrium is always increasing. This does not prevent, however, *local* features of this distribution to take quasi-deterministic values in the limit. Indeed, looking again at Fig. 5, one notices that, numerically,  $\mathbf{A}(x)$  reaches a quasi-deterministic value. As the number of nodes increases, the effect is reinforced (not shown).

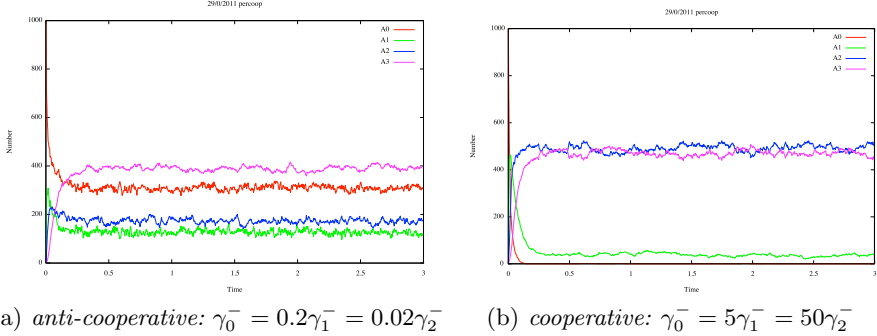
So it is natural to assume constant values for large times for  $\mathbf{A}(x)$  and  $\mathbf{B}(x)$ . The question remains as to how to get these values. One obvious way to do this is by simulation, measuring their average long-term values. But, this limits us to concrete examples. In the next section, assuming  $\Gamma_B$  constant, we show how to derive an equation, the solution of which gives the steady state value of  $\mathbf{A}(x)$ .



**Fig. 3.** A long-term snapshot from the above system with parameters:  $\gamma_{i,j}^- = 2$ ,  $\gamma_{i,j}^+ = 1$ ,  $n_A = 100$ ,  $n_B = 150$ . One sees that the graph alternates. With this set of parameters, we are well past criticality, as we have just one connected site graph.



**Fig. 4.** The total alkenes stays very low with cooperative parameters:  $\gamma_0^- = 5\gamma_1^- = 50\gamma_2^-$ ,  $n_A = 1000$ ,  $n_B = 1500$ ,  $\gamma_{i,j}^+ = 0.1$



**Fig. 5.** Parameters specific to the cooperative and anti-cooperative cases are indicated in the caption. In both cases,  $\gamma^+(i) = 0.1$  is kept constant, and one only varies  $\gamma^-(i)$ , while  $n_A = 1000$ ,  $n_B = 1500$ .

## 4 Deterministic Equilibrium for $\mathbf{A}(x)$

Our next step is to write down a differential equation which describes the average evolution of the  $\mathbf{A}(x)$  and  $\mathbf{B}(x)$ . This is part of a generic process of *fragmentation* of rule-based models described in Ref. [6, 1, 7]. The presentation in 4.1-2 is self-contained. In this simple case, there is a shortcut to the answer, so the reader not interested in the machinery can jump directly to 4.3.

### 4.1 Fragmentation

We have to consider consumption terms caused by the forward rules  $r(i, j)$  and the backward ones  $r^*(i, j)$ , as well as their production terms. We will write simply  $A_i$ ,  $B_j$  (where previously we wrote  $A_i(x)$ ,  $B_j(x)$ ), and  $A_i \circ B_j$  for the ‘bond observable’, namely a pair of an  $A_i$  and a  $B_j$  joined by a (unique) bond. This implies  $i, j > 0$ , as in our convention, indexes  $i, j$  include the bond itself (eg,  $A_1 \circ B_1$  denotes a bond between an  $A$  and a  $B$  which have no other bond).

We can decompose the event horizon of  $x \in \mathcal{G}(n_A, n_B)$  (that is to say the set of all events that are possible at  $x$ ) depending on:

- the rule the event is a event of, and
- whether the event increases/decreases the number of  $A_i$ s (indicated by a  $\pm$  sign below).

The second column in the tables below records the count of the total number of consumption/productions pairs where an  $A_i$  occurrence and an event intersect, meaning that the event modifies the said occurrence. Note that by convention here we produce/consume occurrences (events are defined as in 2’s definition of applying a rule) not matchings, as this is simpler in our symmetric setting.

For the consumptions:

Rule	# collision pairs	conditions on $i, j$
$r(i, j)$	$(v_A - i)(v_B - j)A_i B_j$	$0 \leq i, j < v_A, v_B$
$r^*(i - 1, j - 1)$	$A_i \circ B_j$	$0 < i, j \leq v_A, v_B$

For the productions:

Rule	# production pairs	conditions on $i, j$
$r(i - 1, j - 1)$	$(v_A - i + 1)(v_B - j + 1)A_{i-1} B_{j-1}$	$0 < i, j \leq v_A, v_B$
$r^*(i, j)$	$A_{i+1} \circ B_{j+1}$	$0 \leq i, j < v_A, v_B$

As the number of collision/production pairs involving  $A_i$  multiplied by the rule rate constant gives the average number of modifications of  $A_i$  per time unit, we can write the general differential equation for  $A_i$ :

$$\begin{aligned}
A'_i = & -\mathbf{1}_{i < v_A} \sum_{0 \leq j < v_B} \gamma_{i,j}^+ (v_A - i)(v_B - j) A_i B_j \\
& - \mathbf{1}_{i > 0} \sum_{0 < j \leq v_B} \gamma_{i-1,j-1}^- A_i \circ B_j \\
& + \mathbf{1}_{i > 0} \sum_{0 < j \leq v_B} \gamma_{i-1,j-1}^+ (v_A - i + 1)(v_B - j + 1) A_{i-1} B_{j-1} \\
& + \mathbf{1}_{i < v_A} \sum_{0 \leq j < v_B} \gamma_{i,j}^- A_{i+1} \circ B_{j+1}
\end{aligned} \tag{6}$$

We get similar equations for the  $B_j$ s, but we are not interested in those.<sup>3</sup>

## 4.2 Additional Assumption

The problem with the equations above is that they do not form a *self-consistent* system. Some terms require the knowledge of the ‘bond observables’  $A_i \circ B_j$  which one can *a priori* only infer from the knowledge of  $x$ . Following the general fragmentation procedure at this stage, would need writing a similar ODE for the  $A_i \circ B_j$ s. This is possible. But the problem is that one can now have an event centered on a given bond between  $u, v$  of type  $A, B$  (whether to create it or to delete it) which intersects other bond observables attached to  $u$  and  $v$ . Thus, the time derivative of  $A_i \circ B_j$  will contain ternary terms of the form  $A_i \circ B_j \circ A_k$ , etc. It is easy to show that this procedure generates unboundedly many observables. So if we want some handle on the eventual mean values of  $\mathbf{A}, \mathbf{B}$ , we need to do something else.

Suppose  $\gamma_{i,j}^- = \gamma_i^-$ , and  $\gamma_{i,j}^+ = \gamma_i^+$  do not depend on  $j$ , we can rewrite the ODE above self-consistently:

$$\begin{aligned}
A'_i = & \mathbf{1}_{\{i < v_A\}} \cdot (\gamma_i^- (i + 1) A_{i+1} - \gamma_i^+ (v_A - i) A_i n_b^f) \\
& + \mathbf{1}_{\{i > 0\}} \cdot (-\gamma_{i-1}^- i A_i + \gamma_{i-1}^+ (v_A - i + 1) A_{i-1} n_b^f)
\end{aligned} \tag{7}$$

<sup>3</sup> For the reader familiar with fragmentation, as the graph is bipartite there is no approximation in expressing  $[A_i, B_j; x]$  as  $[A_i; x] \times [B_j; x]$ .

where we have written  $n_b^f$  for the the *number of free sites of type b* in  $x$  (not to be confused with the number of free  $B$ 's, written  $B_0$ ) which satisfies:

$$n_b^f = v_B n_B - \sum_{0 \leq j \leq v_B} j B_j = v_A n_A - \sum_{0 \leq i \leq v_A} i A_i \quad (8)$$

This simplified differential system, indexed by  $0 \leq i \leq v_A$  is now indeed self-consistent as every variable, including  $n_b^f$  which using Eq.8 can also be expressed in terms of the  $A_i$ s, and thus, there is no longer a need to look back at  $x$  to understand the implied dynamics (even the  $B_j$ s are not needed).

To see in details how the terms of the equation above simplify, consider the following:

$$\begin{aligned} \sum_{0 \leq j < v_B} \gamma_{i,j}^+(v_A - i)(v_B - j) A_i B_j &= \gamma_i^+(v_A - i) A_i n_b^f \\ \sum_{0 < j \leq v_B} \gamma_{i-1,j-1}^- A_i \circ B_j &= \gamma_{i-1}^- \sum_{0 < j \leq v_B} A_i \circ B_j = \gamma_{i-1}^- i A_i \\ \sum_{0 < j \leq v_B} \gamma_{i-1,j-1}^+(v_A - i + 1)(v_B - j + 1) A_{i-1} B_{j-1} &= \gamma_{i-1}^+(v_A - i + 1) A_{i-1} n_b^f \\ \sum_{0 \leq j < v_B} \gamma_{i,j}^- A_{i+1} \circ B_{j+1} &= \gamma_i^- \sum_{0 \leq j < v_B} A_{i+1} \circ B_{j+1} = \gamma_i^-(i + 1) A_{i+1} \end{aligned}$$

Note that our new assumption, which will hold onwards,  $\gamma_{i,j}^\pm = \gamma_i^\pm$ , evidently entails the equilibrium existence condition of Th.1. It would be enough to suppose  $\gamma_{i,j}^- = \gamma_i^-$  to obtain self-consistency, but with a more complex ODE, which does not look very tractable. It is worth noting that our additional assumption breaks the symmetry between  $A$ s and  $B$ s; in particular, we are not able to write *at the same time* a self-consistent ODE system for the  $B_j$ s, unless we suppose that  $\gamma_{i,j}^-$  does not depend on  $i$  either (which takes us back to the simpler case treated in Ref. [4]).

### 4.3 Deterministic Steady State

Putting everything together we obtain:

**Proposition 3.** *The deterministic steady state value of  $A_i$ ,  $0 \leq i \leq v_A$ , assuming  $\gamma_{i,j}^\pm = \gamma_i^\pm$ , is given by the following system of equations with parameters  $n_A$ ,  $n_B$ , and  $\Gamma_i$ ,  $0 \leq i \leq v_A$ :*

$$\begin{aligned} n_A &= \sum_{0 \leq i \leq v_A} A_i \\ n_b^f &= v_B n_B - \sum_{0 \leq i \leq v_A} i A_i \\ A_i &= \binom{v_A}{i} \prod_{0 \leq k < i} \Gamma_k^{-1} \cdot (n_b^f)^i \cdot A_0 \end{aligned} \quad (9)$$



*Proof.* If we write  $\alpha_i^- = \gamma_i^-(i+1)$ , and  $\alpha_i^+ = \gamma_i^+(v_A - i)n_b^f$ , and set  $A_i' = 0$  in Eq.7, we get the following systems of equations:

$$\begin{aligned}\alpha_0^- A_1 &= \alpha_0^+ A_0 \\ \alpha_{i-1}^- A_i + \alpha_i^+ A_i &= \alpha_i^- A_{i+1} + \alpha_{i-1}^+ A_{i-1} \\ \alpha_{v_A-1}^- A_{v_A} &= \alpha_{v_A-1}^+ A_{v_A-1}\end{aligned}$$

which implies  $A_{i+1} = \alpha_i^+ / \alpha_i^- A_i = (v_A - i) / (i+1) \cdot \Gamma_i^{-1} \cdot n_b^f \cdot A_i$  for  $0 \leq i < v_A$ .  $\square$

Note that there is a *shortcut* to derive the above equations, by saying directly that, at equilibrium, one must have  $\gamma_i^-(i+1)A_{i+1} = \gamma_i^+(v_A - i)n_b^f A_i$ , that is to say the likelihood that an  $A_{i+1}$ -link breaks has to equal the likelihood that one is created.

We can rescale this equations by making all quantities relative to the total node population  $N$ ; this needs modifying the ‘volume’ of the system and replacing  $\Gamma_i$  with  $\Gamma_i/N$ .

Writing  $a = n_A/N$ ,  $b = n_B/N$ ,  $a_i = A_i/N$  and:

$$K_i := \frac{\prod_{0 \leq k < i} \Gamma_k/N}{\binom{v_A}{i}}$$

we obtain the scale-less steady state equations, with unknowns  $a_i$ , for  $0 < i \leq v_A$ :

$$K_i \cdot a_i = \left( a - \sum_{0 < k \leq v_A} a_k \right) \cdot \left( v_B b - \sum_{0 \leq k \leq v_A} k a_k \right)^i \quad (10)$$

The solutions  $a_i$  give us the mean long-term values  $A_i = N a_i$ , which is what we wanted. We cannot solve this equation in closed form, but we can solve them much faster numerically than by averaging many simulations.

## 5 Criticality

We would like now to understand under which conditions a giant cluster will appear at equilibrium with high probability (that is to say with a probability that tends to 1 as  $N \rightarrow \infty$ ).

As said earlier, graphs in  $\mathcal{G}(n_A, n_B)$  with the same  $a_i$  statistics are equally likely. As we assume that these  $a_i$ s are the solutions to Eq.10 above (deterministic approximation), we can interpret  $a_i$  as the probability that a given node of type  $A$  has  $i$  neighbours, and we are in the familiar situation of a fixed degree distribution random graph model (aka Molloy-Reed [11,5]); except for the slight simplification that our graphs are all bipartite, so there is no need to condition out self-loops (we still have multiple edges or ‘alkenes’ which we neglect).

Let us write  $\mu$  for  $\sum i a_i = \sum j b_j$ . We can think of the exploration of a connected component in terms of a branching process. We start from a bound  $b$ ,

which is linked to an  $A_i$  with probability  $\alpha_i$ , and therefore gives us  $(i - 1)$  more bound  $as$  to follow (because we just used one); each of these  $as$  give a  $B_j$  with probability  $\beta_j$ , and so  $(j - 1)$  new bound  $bs$ .

Now  $\alpha_i$  is proportional to  $iA_i$ , because an  $A$  node is picked with a probability proportional to its degree. In other words,  $\alpha_i := iA_i / \sum_i iA_i = ia_i / \mu$ , and, likewise,  $\beta_j := jB_j / \sum_j jB_j = jb_j / \mu$ . So the mean number of bound  $b$  sites discovered, starting with one, and completing one cycle is given by:

$$\begin{aligned} \nu &= \sum_{i \leq v_A, j \leq v_B} a_i b_j i(i-1)j(j-1) / \mu^2 \\ &= \sum_{i \leq v_A} a_i i(i-1) / \mu \cdot \sum_{j \leq v_B} b_j j(j-1) / \mu \end{aligned}$$

Introducing  $p(A_i) = Na_i/n_A$ ,  $p(B_j) = Nb_j/n_B$ , the probabilities that an  $A$  has degree  $i$ , and a  $B$  degree  $j$ , one can rewrite the above as:<sup>4</sup>

$$\begin{aligned} \nu &= \frac{N^2}{\mu^2 n_A n_B} \sum_{i \leq v_A} p(A_i) i(i-1) \cdot \sum_{j \leq v_B} p(B_j) j(j-1) \\ &= \frac{N^2}{n_A n_B} \frac{\theta_A \theta_B}{\mu^2} \end{aligned}$$

where  $\theta_A, \theta_B$  are the so-called second *factorial moments* of the  $A$  and  $B$  degree distributions.

Whenever a branching law has finite mean, we know by a classical result [8, Th 6.1 8.1] that: 1) there is a non-zero probability for the associated branching process to never extinguish iff  $\nu > 1$ , and 2) the population in the  $k$ th generation, here the number of bound  $b$  sites, grows geometrically as  $\nu^k$ . This means that, asymptotically, the exploration described above will discover an infinite cluster if  $\nu > 1$ . So the criticality condition reads (using notations  $a = n_A/N$ ,  $b = n_B/N$  introduced earlier in 4.3):

$$\theta_A \theta_B > \mu^2 ab \quad (11)$$

Observe that all quantities involved are scale-less and have an interesting probabilistic interpretation. First,  $a$  and  $b$  are the probabilities that a node has respective types  $A$  and  $B$ , so  $a + b = 1$  and  $ab$  is the variance of the ‘composition’ of the initial state. The higher it is, the less critical the system. Second,  $\mu$  is half the mean degree. Finally,  $\theta_A, \theta_B$  are variance-like quantities that measure the noise on their respective degree distributions.

Be that as it may, using Eq.10, ie numerically solving it, we can compute  $\theta_A$  from the  $a_i$ s. This also gives the total number of links, hence  $\sum_i iB_i$ . This is all we need to use the estimate of 2.4, and compute  $\theta_B$ . Indeed, the estimate says that  $B$  has degree  $q$  with probability  $1 - \rho$ , and degree  $q + 1$  with probability  $\rho$ , with  $\sum_i iB_i = (q + \rho)n_B$ . In which case  $q(q - 1) \leq \theta_B = q(q - 1 + 2\rho) \leq q(q + 1)$ , with  $q$  the integer part (rounded below) of the average degree  $\sum_i iB_i/n_B = \mu/b$  of a node of type  $B$ .

<sup>4</sup> Earlier in 2 we have written simply  $p_i$  for  $p(B_i)$ .

## References

1. Danos, V., Feret, J., Fontana, W., Harmer, R., Krivine, J.: Abstracting the differential semantics of rule-based models: Exact and automated model reduction. In: Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, Edinburgh, United Kingdom, July 11-14, pp. 362–381 (2010)
2. Danos, V., Feret, J., Fontana, W., Krivine, J.: Abstract interpretation of cellular signalling networks. In: Logozzo, F., Peled, D.A., Zuck, L.D. (eds.) VMCAI 2008. LNCS, vol. 4905, pp. 83–97. Springer, Heidelberg (2008)
3. Danos, V., Oury, N.: Equilibrium and termination II: the case of Petri nets (2010) (to appear in MSCS)
4. Danos, V., Schumacher, L.J.: How liquid is biological signalling? Theoretical Computer Science 410(11), 1003–1012 (2009)
5. Durrett, R.: Random graph dynamics, vol. 20. CUP (2007)
6. Feret, J., Danos, V., Harmer, R., Krivine, J., Fontana, W.: Internal coarse-graining of molecular systems. PNAS 106(16), 6453–6458 (2009)
7. Harmer, R., Danos, V., Feret, J., Krivine, J., Fontana, W.: Intrinsic Information carriers in combinatorial dynamical systems. Chaos 20(3), 037108 (2010)
8. Harris, T.: The theory of branching processes. Dover (1989)
9. Koepl, H., Hafner, M., Danos, V.: Rule-based modeling for protein-protein interaction networks - the Cyanobacterial circadian clock as a case study. In: WCSB 2009 Proceedings, pp. 87–91 (2009)
10. Koepl, H., Schumacher, L.J., Danos, V.: A statistical analysis of receptor clustering using random graphs. In: WCSB 2009 Proceedings, pp. 95–99 (2009)
11. Molloy, M., Reed, B.: A critical point for random graphs with a given degree sequence. Random Structures & Algorithms 6(2-3), 161–180 (1995)
12. Ollivier, J., Shahrezaei, V., Swain, P.: Scalable rule-based modelling of allosteric proteins and biochemical networks. PLoS Computational Biology 6(11) (2010)
13. Qian, L., Winfree, E.: Scaling up digital circuit computation with dna strand displacement cascades. Science 332(6034), 1196 (2011)
14. Saiz, L., Vilar, J.: Stochastic dynamics of macromolecular-assembly networks. Molecular Systems Biology 2(1) (2006)
15. Sneddon, M., Faeder, J., Emonet, T.: Efficient modeling, simulation and coarse-graining of biological complexity with nfsim. Nature Methods (2010)
16. Williamson, J.R.: Cooperativity in macromolecular assembly. Nat. Chem. Biol. 4(8), 458–465 (2008)

## A Distinguishing Sites - Compilation to Kappa

The model described below uses the KaSim simulator. (For a general and concise example of rule-based modelling, a possible entry point is Ref. [9].) There are other rule-based simulators such as the recent NFsim which one could use to do this just as well [15].

As KaSim does not know indistinguishable sites, in order to experiment numerically with our models, we need to understand how to design equivalent ones where all sites are locally distinguished. This is an interesting question in its own right, which we deal with now.

For the purpose of the discussion we will call site graphs where sites of a given node have distinct types *concrete*, and those where, as we have done so far, we allow many sites of a same node to be of the same type, we will call *abstract*. Two concrete graphs which only differ by permutations of sites local to nodes will be said to be *equivalent*. This defines an equivalence relation on concrete site graphs, the class of which are the abstract ones.

For definiteness, in our concrete graphs, we will choose  $(a_1, \dots, a_{v_A})$  for the sites of nodes of type  $A$ , and  $(b_1, \dots, b_{v_B})$  for those of nodes of type  $B$ .

In order to define the dynamics on concrete site graphs, we replace the abstract reversible rule  $r(i, j)$  with a *set*  $R(i, j)$  of concrete reversible rules which enumerate all possibilities in the following way. We choose first a pair of a free  $a$  and a free  $b$  to be bound by the rule, and then, among the remaining sites, we choose the  $i$  bound  $as$ , the  $j$  bound  $bs$ .

Thus  $|R(i, j)| = v_A v_B \cdot \binom{v_A-1}{i} \binom{v_B-1}{j}$ , where the  $-1$  in the ‘choose’ terms comes from the fact that we already have chosen one  $a$  or  $b$ , namely the one we want to bind in the concrete rule of interest.

Note that  $\sum_{i, j < v_A, v_B} |R(i, j)| = v_A v_B 2^{v_A-1} 2^{v_B-1}$ , so, unsurprisingly, the concrete rule set is exponentially larger than the original one,  $v_A v_B$ , in the valence of  $A$  and  $B$ .

Let a pair of unconnected nodes with degrees  $i < v_A, j < v_B$  be given in a concrete site graph. Exactly  $(v_A - i)(v_B - j)$  forward rules in  $R(i, j)$  apply to this pair, and they apply uniquely (ie with a unique match). Indeed the only choice remaining in a concrete site graph, is that of the free sites. All choices lead to different concrete graphs, but the same abstract one. Similarly, given a pair of connected nodes with degrees  $i \leq v_A, j \leq v_B$ , exactly one backward rule in  $R(i, j)$  applies, namely the one that breaks the (unique) connexion between the nodes.

Pick  $x, y$  abstract site graphs connected via an  $r(i, j)$  transition, and a concrete  $c$  above  $x$ . By the considerations above, the rate at which  $c$  jumps in  $y$  (meaning to some concrete graph in the equivalence class  $y$ ) is  $\gamma^+(i, j)(v_A - i)(v_B - j)$ , while the reverse rate at which some  $c'$  over  $y$  jumps in  $x$  is  $\gamma^-(i, j)$ .

Hence, the dynamics of the concrete transition system is identical to the one we have used in our theoretical investigation (one says sometimes that the quotient is a strong stochastic bisimulation of continuous-time Markov chains), and we can safely use it to explore the behaviour of our models in a numerical way. We can now turn to the description of our concrete model.

## B The Numerical Model

Our working model can be obtained here, the simulator KaSim here. The specific instance below is parametrized as is the cooperative model run in Fig. 5(b).

### B.1 Agents, Parameters and Initial State

We start by declaring agents and the main parameters, including a fictitious volume parameter to be able to rescale simulations easily.

```

%agent: B(b1,b2)
%agent: A(a1,a2,a3)

%var: 'vol' 100
%var: 'k_on' 0.1/'vol'
%var: 'k_off' 2
%var: 'k_off_vee' 1/5 * 'k_off'
%var: 'k_off_tee' 1/50 * 'k_off'
%var: 'n_A' (1000 * 'vol')
%var: 'n_B' (1500 * 'vol')

```

Then we define the initial state (written  $x_0$  earlier).

```

%init: 'n_A' (A(a1,a2,a3))
%init: 'n_B' (B(b1,b2))

```

## B.2 Rules

Now we need the interaction rules. From the preceding subsection, and since we have assumed that rules have no dependency in the degree of  $B$ , we expect to have to write  $v_A v_B 2^{v_A-1} = 24$  reversible rules (so 48 KaSim rules). Note that all rates uses the parameters defined earlier for more flexibility.

```

'b1-a1-11' B(b1), A(a1,a2!_,a3!_) -> B(b1!0), A(a1!0,a2!_,a3!_)@ 'k_on'
'b1-a1-10' B(b1), A(a1,a2!_,a3 ) -> B(b1!0), A(a1!0,a2!_,a3 )@ 'k_on'
'b1-a1-01' B(b1), A(a1,a2 ,a3!_) -> B(b1!0), A(a1!0,a2 ,a3!_)@ 'k_on'
'b1-a1-00' B(b1), A(a1,a2 ,a3 ) -> B(b1!0), A(a1!0,a2 ,a3 )@ 'k_on'

'b1 a1-11' B(b1!0), A(a1!0,a2!_,a3!_) -> B(b1), A(a1,a2!_,a3!_)@ 'k_off_tee'
'b1 a1-10' B(b1!0), A(a1!0,a2!_,a3 ) -> B(b1), A(a1,a2!_,a3 )@ 'k_off_vee'
'b1 a1-01' B(b1!0), A(a1!0,a2 ,a3!_) -> B(b1), A(a1,a2 ,a3!_)@ 'k_off_vee'
'b1 a1-00' B(b1!0), A(a1!0,a2 ,a3 ) -> B(b1), A(a1,a2 ,a3 )@ 'k_off'

'b1-a2-11' B(b1), A(a2,a1!_,a3!_) -> B(b1!0), A(a2!0,a1!_,a3!_)@ 'k_on'
'b1-a2-10' B(b1), A(a2,a1!_,a3 ) -> B(b1!0), A(a2!0,a1!_,a3 )@ 'k_on'
'b1-a2-01' B(b1), A(a2,a1 ,a3!_) -> B(b1!0), A(a2!0,a1 ,a3!_)@ 'k_on'
'b1-a2-00' B(b1), A(a2,a1 ,a3 ) -> B(b1!0), A(a2!0,a1 ,a3 )@ 'k_on'

'b1 a2-11' B(b1!0), A(a2!0,a1!_,a3!_) -> B(b1), A(a2,a1!_,a3!_)@ 'k_off_tee'
'b1 a2-10' B(b1!0), A(a2!0,a1!_,a3 ) -> B(b1), A(a2,a1!_,a3 )@ 'k_off_vee'
'b1 a2-01' B(b1!0), A(a2!0,a1 ,a3!_) -> B(b1), A(a2,a1 ,a3!_)@ 'k_off_vee'
'b1 a2-00' B(b1!0), A(a2!0,a1 ,a3 ) -> B(b1), A(a2,a1 ,a3 )@ 'k_off'

'b1-a3-11' B(b1), A(a3,a1!_,a2!_) -> B(b1!0), A(a3!0,a1!_,a2!_)@ 'k_on'
'b1-a3-10' B(b1), A(a3,a1!_,a2 ) -> B(b1!0), A(a3!0,a1!_,a2 )@ 'k_on'
'b1-a3-01' B(b1), A(a3,a1 ,a2!_) -> B(b1!0), A(a3!0,a1 ,a2!_)@ 'k_on'
'b1-a3-00' B(b1), A(a3,a1 ,a2 ) -> B(b1!0), A(a3!0,a1 ,a2 )@ 'k_on'

'b1 a3-11' B(b1!0), A(a3!0,a1!_,a2!_) -> B(b1), A(a3,a1!_,a2!_)@ 'k_off_tee'
'b1 a3-10' B(b1!0), A(a3!0,a1!_,a2 ) -> B(b1), A(a3,a1!_,a2 )@ 'k_off_vee'
'b1 a3-01' B(b1!0), A(a3!0,a1 ,a2!_) -> B(b1), A(a3,a1 ,a2!_)@ 'k_off_vee'
'b1 a3-00' B(b1!0), A(a3!0,a1 ,a2 ) -> B(b1), A(a3,a1 ,a2 )@ 'k_off'

```

```

'b2-a1-11' B(b2), A(a1,a2!_,a3!_) -> B(b2!0), A(a1!0,a2!_,a3!_)@ 'k_on'
'b2-a1-10' B(b2), A(a1,a2!_,a3 ) -> B(b2!0), A(a1!0,a2!_,a3 )@ 'k_on'
'b2-a1-01' B(b2), A(a1,a2 ,a3!_) -> B(b2!0), A(a1!0,a2 ,a3!_)@ 'k_on'
'b2-a1-00' B(b2), A(a1,a2 ,a3 ) -> B(b2!0), A(a1!0,a2 ,a3 )@ 'k_on'

'b2 a1-11' B(b2!0), A(a1!0,a2!_,a3!_) -> B(b2), A(a1,a2!_,a3!_)@ 'k_off_tee'
'b2 a1-10' B(b2!0), A(a1!0,a2!_,a3 ) -> B(b2), A(a1,a2!_,a3 )@ 'k_off_vee'
'b2 a1-01' B(b2!0), A(a1!0,a2 ,a3!_) -> B(b2), A(a1,a2 ,a3!_)@ 'k_off_vee'
'b2 a1-00' B(b2!0), A(a1!0,a2 ,a3 ) -> B(b2), A(a1,a2 ,a3 )@ 'k_off'

'b2-a2-11' B(b2), A(a2,a1!_,a3!_) -> B(b2!0), A(a2!0,a1!_,a3!_)@ 'k_on'
'b2-a2-10' B(b2), A(a2,a1!_,a3 ) -> B(b2!0), A(a2!0,a1!_,a3 )@ 'k_on'
'b2-a2-01' B(b2), A(a2,a1 ,a3!_) -> B(b2!0), A(a2!0,a1 ,a3!_)@ 'k_on'
'b2-a2-00' B(b2), A(a2,a1 ,a3 ) -> B(b2!0), A(a2!0,a1 ,a3 )@ 'k_on'

'b2 a2-11' B(b2!0), A(a2!0,a1!_,a3!_) -> B(b2), A(a2,a1!_,a3!_)@ 'k_off_tee'
'b2 a2-10' B(b2!0), A(a2!0,a1!_,a3 ) -> B(b2), A(a2,a1!_,a3 )@ 'k_off_vee'
'b2 a2-01' B(b2!0), A(a2!0,a1 ,a3!_) -> B(b2), A(a2,a1 ,a3!_)@ 'k_off_vee'
'b2 a2-00' B(b2!0), A(a2!0,a1 ,a3 ) -> B(b2), A(a2,a1 ,a3 )@ 'k_off'

'b2-a3-11' B(b2), A(a3,a1!_,a2!_) -> B(b2!0), A(a3!0,a1!_,a2!_)@ 'k_on'
'b2-a3-10' B(b2), A(a3,a1!_,a2 ) -> B(b2!0), A(a3!0,a1!_,a2 )@ 'k_on'
'b2-a3-01' B(b2), A(a3,a1 ,a2!_) -> B(b2!0), A(a3!0,a1 ,a2!_)@ 'k_on'
'b2-a3-00' B(b2), A(a3,a1 ,a2 ) -> B(b2!0), A(a3!0,a1 ,a2 )@ 'k_on'

'b2 a3-11' B(b2!0), A(a3!0,a1!_,a2!_) -> B(b2), A(a3,a1!_,a2!_)@ 'k_off_tee'
'b2 a3-10' B(b2!0), A(a3!0,a1!_,a2 ) -> B(b2), A(a3,a1!_,a2 )@ 'k_off_vee'
'b2 a3-01' B(b2!0), A(a3!0,a1 ,a2!_) -> B(b2), A(a3,a1 ,a2!_)@ 'k_off_vee'
'b2 a3-00' B(b2!0), A(a3!0,a1 ,a2 ) -> B(b2), A(a3,a1 ,a2 )@ 'k_off'

```

### B.3 Observables

Finally we need to define observables, so that the simulator understands what to plot. Here we observe **A**; **B** would be done similarly.

```

%var: 'A0' A(a1,a2,a3)

%var: 'a1' A(a1!_,a2,a3)
%var: 'a2' A(a1,a2!_,a3)
%var: 'a3' A(a1,a2,a3!_)

%var: 'A1' 'a1' + 'a2' + 'a3'

%var: 'a1a2' A(a1!_,a2!_,a3)
%var: 'a1a3' A(a1!_,a2,a3!_)
%var: 'a2a3' A(a1,a2!_,a3!_)

%var: 'A2' 'a1a2' + 'a1a3' + 'a2a3'

%var: 'A3' A(a1!_,a2!_,a3!_)

```

```

%plot: 'A0'
%plot: 'A1'
%plot: 'A2'
%plot: 'A3'

```

Finally, we observe 'alkenes' to get a sense of their frequency (very low as we have seen in the main text).

```

%var: 'Ba1a2' B(b1!1,b2!2),A(a1!1,a2!2)
%var: 'Ba2a1' B(b1!1,b2!2),A(a1!2,a2!1)
%var: 'Ba1a3' B(b1!1,b2!2),A(a1!1,a3!2)
%var: 'Ba3a1' B(b1!1,b2!2),A(a1!2,a3!1)
%var: 'Ba2a3' B(b1!1,b2!2),A(a2!1,a3!2)
%var: 'Ba3a2' B(b1!1,b2!2),A(a2!2,a3!1)

%var: 'alkene' 'Ba1a2'+ 'Ba2a1' + 'Ba1a3' + 'Ba3a1' + 'Ba2a3' + 'Ba3a2'

%plot: 'alkene'

```

This completes the definition of our example model. Each of the above modules can be written as a separate file which again increases clarity and flexibility.

# The Computer Science of Molecular Programming

Jack H. Lutz\*

Department of Computer Science  
Iowa State University  
Ames, IA 50011, USA  
`lutz@cs.iastate.edu`

**Abstract.** Computer science has its historical roots in mathematical logic and electrical engineering. However it quickly evolved into a separate discipline with its own methods for describing and controlling aspects of reality not addressed by its predecessors. This talk will examine ways in which computer science concepts—including abstraction, modularity, state, universality, concurrency, safety, specifications, verification, complexity, and randomness—are contributing to the development of molecular programming.

---

\* This research was supported in part by National Science Foundation Grant 0652569.



# An Autonomous DNA Nanodevice Captures pH Maps of Living Cells in Culture and *in Vivo*

Sunaina Surana, Souvik Modi, and Yamuna Krishnan\*

National Centre for Biological Sciences, Tata Institute of Fundamental Research,  
UAS-GKVK, Bellary Road, Bangalore: 560065, India  
yamuna@ncbs.res.in

**Abstract.** DNA nanomachines are assemblies that rely on molecular inputs that are processed or transduced into measurable outputs. Though DNA nanotechnology has created a gamut of molecular devices, an outstanding challenge has been the demonstration of functionality and relevance of these devices in living systems. The I-switch is a DNA nanodevice that, in response to protons, changes its conformation to produce a fluorescence resonance energy transfer (FRET) signal. We show that this rationally designed molecular device is capable of measuring spatiotemporal pH changes associated with endosomes as they undergo maturation in living cells in culture. Furthermore, we show that the nanomachine retains its autonomous functionality as it maps the same biological process in cells of a living organism like *C. elegans*. This demonstration of the quantitative functionality of an artificially designed scaffold positions DNA nanodevices as powerful tools to interrogate biological phenomena.

**Keywords:** I-switch, pH sensor, hemocytes, coelomocytes.

## 1 Introduction

Structural DNA nanotechnology utilizes the specificity and versatility of DNA as a building block to artificially fabricate functionally exciting nanoarchitectures [1-3]. These architectures are either self-assembled static scaffolds, or dynamic nanomachines that change conformation in response to external triggers. DNA nanomachines, due to their programmability and ability to respond to a repertoire of environmental cues, possess potential to function as powerful tools to interrogate biological phenomena. Examples include molecular beacons, hybridization-powered molecular motors, DNA walkers and tweezers and aptamer-based switches [1-3]. Despite the existence of a gamut of molecular architectures, functional validation of such artificially designed DNA nanodevices within a biological system has remained elusive.

We describe here the construction and working of an autonomous DNA nanomachine that undergoes a conformational change in response to changes in proton concentration. This nanodevice, called the I-switch, consists of two B-DNA duplexes connected to each other by a hinge, and carrying terminal cytosine-rich single-strand overhangs [4]. On acidification of the medium, the cytosine-rich

---

\* Corresponding author.

overhangs protonate [5] and cause a conformational change in the assembly that is pH-reversible. Opening and closing of the I-switch is monitored by means of fluorescence resonance energy transfer (FRET). Thus, the switch functions as an AND gate requiring two inputs simultaneously to undergo a computation: wavelength of light corresponding to the excitation of the donor fluorophore, and environmental protons, which toggle the I-switch. Presence of both inputs is sufficient to generate an output from the device in the form of wavelength of emission of the acceptor fluorophore. Absence of either fails to elicit a photonic output.

Given the dearth of demonstration of functionality of rationally designed artificial DNA nanomachines outside of the *in vitro* context, the I-switch presents good potential to establish the relevance of synthetic DNA nanodevices in living systems. The biological contexts chosen to study functionality of the I-switch were an *in cellulo* system, namely isolated *Drosophila* hemocytes [6], and an *in vivo* system, coelomocytes of the multicellular model organism *Caenorhabditis elegans* [7]. Both hemocytes and coelomocytes possess scavenging activity and are known to internalize a variety of negatively charged macromolecules from their environment via the anionic ligand binding receptor (ALBR) pathway. This property has led to these cells being used as systems to study endocytosis *in cellulo* and *in vivo*, respectively [6,7]. Hence, we chose endosomal maturation along the ALBR pathway in these two systems to probe functionality of the I-switch. Our studies revealed that this rationally designed DNA nanodevice is internalized by these cells via receptor-mediated endocytosis. Once inside, it retains its autonomous functionality both *in cellulo* and *in vivo* with quantitative precision. It has highest sensitivity in the pH regime  $\sim 5.3 - 6.8$  and is thus capable of mapping spatiotemporal pH changes associated with endosomal maturation [4,8]. This dynamic regime, which coincides well with the pH regimes of most physiological processes, positions the I-switch to track other pH-correlated biological processes in a variety of systems.

## 2 Materials and Methods

### 2.1 Sample Preparation

The I-switch was made by mixing the DNA oligonucleotides O1, O2 and O3 shown in Table 1 in equimolar ratios, heating at 90°C for 5 min, and then slowly cooling to room temperature @ 5°C per 15 minutes. This sample preparation was carried out with oligonucleotide concentrations of 5  $\mu\text{M}$ , in 10 mM phosphate buffer of pH 5.5, in the presence of 100 mM KCl. Samples were equilibrated at 4°C overnight. Fluorescently labeled I-switch was prepared in a similar manner with fluorophore-labeled oligonucleotides.

### 2.2 *In cellulo* Studies

*Drosophila* hemocytes were isolated and maintained from third instar larvae as previously described [6]. Cells were washed with Medium 1 prior to labeling. For pH measurement experiments, cells were imaged live, after chasing the probes for the stated time points. The pH standard curve was generated in cells by briefly fixing cells (for 1 min) with 2.5% paraformaldehyde and then adding 10  $\mu\text{M}$  of the

ionophore nigericin along with Medium 1 buffered to appropriate pH (ranging from 5 - 7). All the wide-field images were acquired using a Nikon inverted microscope equipped with 60 $\times$ , 1.4 NA objectives, a mercury arc illuminator and a cooled CCD camera controlled by Metamorph software. Three sets of images were taken corresponding to (I) image at donor emission wavelength upon donor excitation (donor image), (II) image at acceptor emission wavelength (acceptor FRET) upon donor excitation and (III) image at acceptor emission wavelength (acceptor image) acceptor excitation [4].

**Table 1.** Sequences of oligonucleotides used

Name	Sequence
O1	5'-CCCCAACCCCAATACATTTTACGCCTGGTGCC-3'
O2	5'-CCGACCGCAGGATCCTATAAAACCCCAACCCC-3'
O3	5'-TTATAGGATCCTGCGGTCGGAGGCACCAGGCGTAAAATGTA-3'
O1-488	5'-Alexa 488-CCCCAACCCCAATACATTTTACGCCTGGTGCC-3'
O2-647	5'-CCGACCGCAGGATCCTATAAAACCCCAACCCC-Alexa 647-3'

### 2.3 *In vivo* Studies

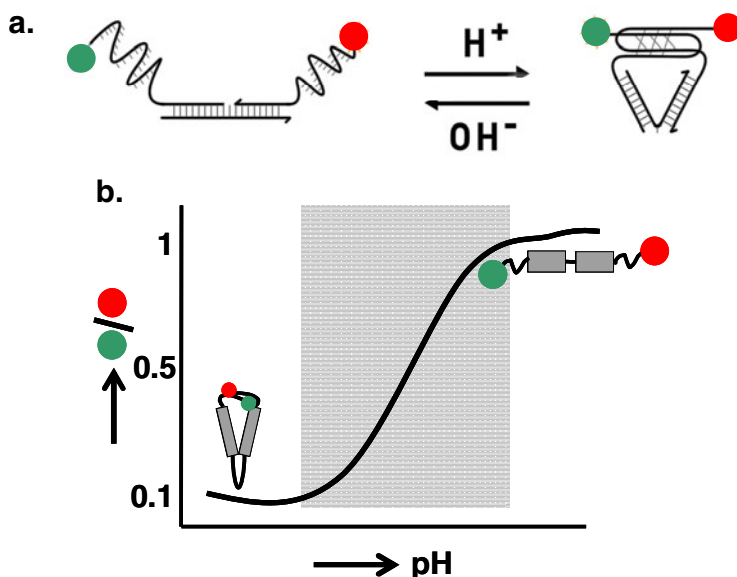
Strains used were the wild type *C. elegans* isolate from Bristol (strain N2), *cdIs131*, *cdIs66* and *pwIs50*. Trafficking of I-switch was determined by performing colocalization studies of GFP and I-switch in transgenics expressing GFP-fusion proteins of endosomal markers. For pH measurements, I-switch was diluted in Medium 1 and microinjected in the body cavity and images acquired after the stated time points. pH was clamped using clamping buffer of the appropriate pH (ranging from 5-7) containing 100  $\mu$ M nigericin and 100  $\mu$ M monensin [8]. Images were acquired on a Nikon inverted microscope as described above.

## 3 Results

### 3.1 Design of the I-Switch

The DNA nanomachine, called the I-switch, has three component DNA oligonucleotides, O1, O2 and O3, where O1 and O2 are hybridized to adjacent sequences on O3, leaving a single base gap in the middle. This base gap acts as the fulcrum of the nanomachine. The pH-responsive elements in this assembly are cytosine-rich tracts present as single-stranded overhangs on the 5' and 3' ends of O1 and O2, respectively [4]. At acidic pH, these cytosines form CH<sup>+</sup>.C non-Watson Crick base pairs, driving the formation of an intramolecular I-motif [5] that yields the 'closed state' of the I-switch. At neutral pH, the I-motif dissociates and entropy as well as electrostatic repulsion between the duplex arms drives the formation of the 'open' linear conformation (Figure 1a). Thus, the I-switch is a second order nanolever that

functions as an AND gate, which requires the presence of protons and donor excitation wavelength of light which is transduced to an output, namely light of wavelength of acceptor emission. The output is monitored by attaching donor (Alexa 488) and acceptor (Alexa 647) fluorophores that undergo fluorescence resonance energy transfer (FRET) to O1 and O2 respectively. Hence, the assembly shows high FRET at pH 5 and low FRET at pH 7. Thus, when a solution of the doubly labeled I-switch ( $I_{A488/A647}$ ) is excited at the donor excitation wavelength (488 nm), the donor/acceptor (D/A) ratio shows a characteristic sigmoidal profile as a function of pH (Figure 1b). Thus, the I-switch can function as a pH sensor in the linear regime of the curve, where each unique D/A value corresponds to a unique pH value.



**Fig. 1.** Schematics showing (a) The structure and working of I-switch. (b) Profile of donor/acceptor (D/A) ratios as a function of increasing pH. The box indicates the pH-sensitive regime of the device.

### 3.2 I-Switch and Endocytosis in *Drosophila* Hemocytes

Cellular processes such as endocytosis show a characteristic change in pH associated with vesicle maturation, that is, as an endosome matures from the early endosome to the lysosome, via the late endosome, it is accompanied by acidification, which is in the regime of  $\sim 6$  to 6.2 in early endosomes, to pH 5.5 in late endosomes and pH 5 in lysosomes [9]. Such acidification is necessary for proper recycling of receptors and trafficking of a ligand and nutrient [10]. Since the pH profile of the I-switch shows good correlation with the pH regime of endosomal maturation, we investigated the capacity of this DNA nanodevice to map pH changes accompanying endosomal maturation first in cultured cells.

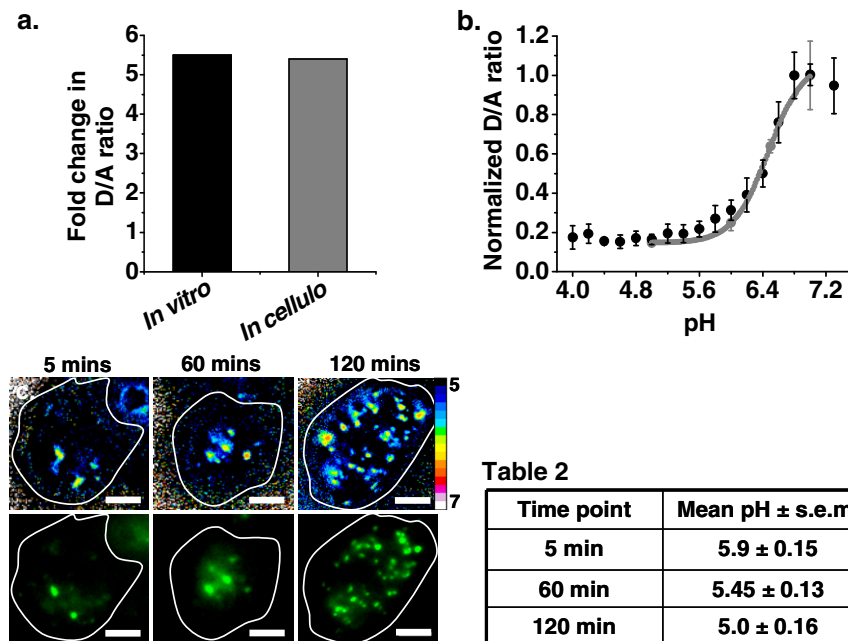
*Drosophila* hemocytes are macrophages that are known to be involved in engulfing a host of foreign molecules. The robust endocytic capabilities of these cells have led to their use as an *in cellulo* system to study endocytic dynamics. Hemocytes possess anionic ligand binding receptors (ALBR), a family of receptors that are known to bind to negatively charged molecules with high affinity and endocytose them [6]. We therefore chose this system to validate the functionality of the I-switch and map spatiotemporal pH changes along the ALBR-mediated endocytic pathway. *Drosophila* hemocytes were incubated with a mixture of 80 nM I-switch labelled with Bodipy TMR ( $I_{\text{BTMR}}$ ) and fluorescein isothiocyanate (FITC)-conjugated dextran (FITC-dextran) ( $1 \text{ mg ml}^{-1}$ ), a marker of the endosomal fluid phase. Importantly, the I-switch was found to be localized in distinct punctate structures  $\sim 1 \mu\text{m}$  in size. When these images were overlaid with co-internalized FITC-dextran images, these puncta were found to co-localize, indicating that the I-switch was indeed marking endosomes.

In order to establish functionality of a sensor, it is first essential to assess its performance quantitatively *in cellulo*. Performance of a sensor, and consequently its integrity, is reflected by the fold change of its donor/acceptor (D/A) ratio in the dynamic regime. Fold change corresponds to the value obtained by dividing the D/A ratio at the higher end of the dynamic regime to that obtained at the lower end (pH 7 and 5, respectively). To measure the fold change *in cellulo* for the I-switch, hemocytes were clamped at pH 5 and pH 7. Cells, pulsed with doubly labelled I-switch ( $I_{\text{A488/A647}}$ ), were fixed and treated with an externally added buffer containing the ionophore nigericin, which equilibrates the intracellular pH to that of the external buffer. After the pH is clamped, the cells were imaged and D/A ratios calculated. The profile of D/A ratios at pH 5 are highly characteristic and different from those at pH 7. Importantly, these profiles yield a fold change of  $\sim 5.4$  from pH 5 to pH 7. This matches very well with the *in vitro* fold change, which is  $\sim 5.5$  in this regime (Figure 2a).

Given that the I-switch retains its performance in terms of its fold change, we proceeded to study its pH-sensitive response over the entire dynamic range. pH was clamped at intermediate values in the dynamic regime, and D/A values plotted as a function of pH. This yields the *in cellulo* pH profile of the I-switch, which shows excellent correspondence with the *in vitro* pH profile (Figure 2b), indicating that the I-switch recapitulates its closing and opening characteristics inside cells both qualitatively and quantitatively.

Since the functionality of the switch in cultured cells has been established, it can now be used to map spatiotemporal pH changes along the endocytic pathway in real time. Cells, pulsed with  $I_{\text{A488/A647}}$ , were chased for 5 min, 1 h and 2 h, which correspond with previously known residence times of ligands in the early endosome, the late endosome and the lysosome, respectively [6]. After imaging, a histogram of D/A ratios of all endosomes revealed that there is gradual acidification of endosomes as they mature (Figure 2c). More importantly, the spread in D/A ratios becomes markedly less from early endosomes to late endosomes to lysosomes, where pH is known to be tightly regulated. These D/A values were then converted into pH values using the standard *in cellulo* calibration curve. These values revealed the pH of the early endosome to be  $\sim 5.9 \pm 0.15$  for the early endosome, pH  $\sim 5.45 \pm 0.13$  for the late endosome and pH  $\sim 5.0 \pm 0.16$  upon maturation to the lysosome (Table 2). These studies also revealed that there is rapid acidification early in the endosomal maturation process where a sharp decrease in D/A is observed over a period of 30 min

followed by a slower decrease over 2 h, suggesting that early endosomes in this pathway rapidly acidify to form the late endosome, which then slowly matures to the lysosome. Thus the I-switch is capable of reporting spatiotemporal pH changes associated with endosome maturation.



**Fig. 2.** Spatio-temporal mapping of endosomal pH changes in living cells in culture using the I-switch. (a) *In vitro* and *in cellulo* fold change in D/A ratios of  $I_{A488/A647}$  at pH 7 to pH 5. (b) Standard calibration curve of  $I_{A488/A647}$  *in cellulo* (grey) and *in vitro* (black) showing normalized D/A ratios versus pH. Error bars indicate s.d. ( $n \geq 35$  endosomes). (c) Pseudocolour D/A map of hemocytes pulsed with  $I_{A488/A647}$  at the indicated chase times. Scale bar: 5  $\mu$ m. Table 2 shows mean endosomal pH ( $\pm$  s.e.m.) at various times in endosomes of the ALBR pathway in hemocytes.

### 3.3 I-Switch and Endocytosis in *Caenorhabditis elegans* Coelomocytes

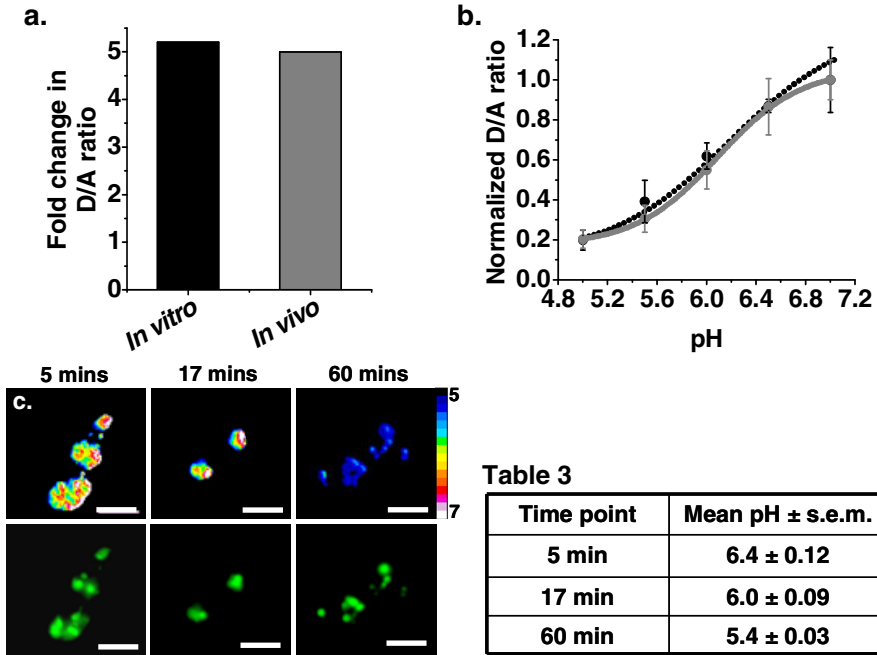
Given the established functionality of the I-switch in cultured cells, we proceeded to investigate its functionality *in vivo*. To this end, we chose the transparent nematode *Caenorhabditis elegans*. This multicellular model organism has six large oblong scavenger cells, called coelomocytes, located in the pseudocoelomic cavity that continuously endocytose macromolecules from the body cavity [7]. Negatively charged macromolecules were shown to be engulfed by the activity of ALBRs present on the surface of coelomocytes [8]. The scavenging activity of these cells has been exploited to study endocytosis in the nematode [7], facilitating the use of these cells as an *in vivo* system to explore the functionality of the present DNA nanomachine.

To study whether the native I-switch marks coelomocytes in *C. elegans*, Alexa 647 labelled I-switch ( $I_{A647}$ ) was injected in the pseudocoelom of 1-day-old wild type

hermaphrodites. After 1 h, it was seen that the I-switch specifically marks a set of six cells, localizing in bright punctate structures of  $\sim 0.8 \mu\text{m}$  size. These puncta were confirmed to be endosomes of the ALBR pathway within coelomocytes [8].

To establish the *in vivo* integrity of the I-switch in coelomocytes, the intra-coelomocyte pH was clamped using clamping buffer containing nigericin and monensin. After imaging, the D/A ratios were calculated and fold change determined. It was found that the *in vivo* fold change ( $\sim 5$ ) also shows remarkable correspondence to the *in vitro* fold change ( $\sim 5.2$ ) (Figure 3a). Furthermore, when the *in vivo* pH calibration curve is generated between pH 5 and pH 7, it matches well with the *in vitro* curve. The *in vivo* pH response curve of the I-switch shows a sigmoidal profile with highest dynamic range in the pH regime 5.3 – 6.6 (Figure 3b). Thus, even on traversing the *in cellulo* to *in vivo* boundary, the integrity of the I-switch is not compromised and it retains its autonomous functionality.

To follow acidification during endosomal maturation of the ALBR pathway, it is necessary to first determine the temporal regimes corresponding to each stage of endosomal maturation along the pathway, that is, determine the residence times of



**Fig. 3.** Spatio-temporal mapping of endosomal pH changes in coelomocytes using the I-switch. (a) *In vitro* and *in vivo* fold change in D/A ratios of  $I_{A488/A647}$  at pH 7 to pH 5. (b) pH calibration curve of  $I_{A488/A647}$  *in vivo* (grey) and *in vitro* (black) showing normalized D/A ratios versus pH. Error bars indicate s.e.m. ( $n \geq 50$  endosomes). (c) Representative pseudocolour D/A images of  $I_{A488/A647}$  labeled coelomocytes in wild type hermaphrodites at indicated times. Scale bar: 10  $\mu\text{m}$ . Table 3 indicates mean endosomal pH ( $\pm$  s.e.m.) at various times in coelomocytes of wild type hermaphrodites.

internalized I-switch in the early endosome, the late endosome and the lysosome. Therefore, time-course experiments were performed with  $I_{A647}$  in hermaphrodites expressing green fluorescent protein (GFP) fused to well-known endocytic compartment markers in the coelomocytes. Markers employed for this purpose were RAB-5 for the early endosome, RAB-7 for the late endosome/lysosome and LMP-1 for the lysosome [11-13]. Injection of  $I_{A647}$  in GFP:: $RAB-5$  expressing transgenic showed that the I-switch is resident in the early endosome maximally at 5 min. After 15 min, the early endosomes quickly mature into late endosomes, as evidenced by the steep drop in co-localization between the I-switch and GFP:: $RAB-5$  and increase in co-localization between GFP:: $RAB-7$ . By 60 min, ~90% of the I-switch is resident in the lysosome, showing high co-localization with LMP-1::GFP. Thus, the 5 min, 17 min and 60 min time points were chosen for pH measurements in the early endosome, the late endosome and the lysosome, respectively.

Next,  $I_{A488/A647}$  was injected in wild type hermaphrodites, and pH measurements made at the stated time points. D/A ratios of a population of endosomes at each stage show gradual acidification with each step. Notably, the decrease in the spread in D/A ratios and thus pH, as we proceed from the early endosomes to the late endosomes and then on to the lysosomes, is seen here as well (Figure 3c). This is consistent with other studies in the literature that have measured pH heterogeneity at different endosomal stages [9], as well as *in cellulo* measurements of the same with the I-switch. These D/A ratios indicate a mean pH of  $\sim 6.4 \pm 0.12$  in the early endosome,  $\sim 6.0 \pm 0.09$  in the late endosome and  $\sim 5.4 \pm 0.03$  in the lysosome (Table 3).

## 4 Discussion

We demonstrate the successful operation of an artificially designed DNA nanomachine inside living cells in culture as well as within a living organism without compromising its efficiency. It recapitulates its sensing properties qualitatively as well as quantitatively *in cellulo* as well as *in vivo*. Given the high fidelity of the overall pH performance of the I-switch in cultured cells as well as *in vivo*, it is possible to effectively map spatiotemporal pH changes associated with endosomal maturation of the ALBR-mediated pathway that it marks in hemocytes and coelomocytes.

Significantly, because the by-products of one cycle of the nanomachine are water and salt, it is non-toxic and does not perturb its own processivity, as well as the living system under investigation. The I-switch has highest pH sensitivity between pH 5.5 and 6.8 and offers complementary information to that obtained through the use of small-molecule fluorescent pH probes. The I-switch is a FRET-based sensor that is ratiometric and utilizes photostable, bright fluorophores. Most importantly, with conventional pH probes based on GFP [14] or small molecules [15] one is limited by a fixed wavelength, but with the I-switch, which is an artificially designed DNA scaffold, one can incorporate fluorophores of any wavelength that form an appropriate FRET pair. It can therefore be used to track multiple proteins tagged with fluorescent proteins. This property thus positions it as a powerful probe to study events in a variety of fluorescent protein expressing backgrounds. Its autonomous functionality also ensures that it can be employed in a variety of mutant backgrounds, thus helping to elucidate molecular and genetic pathways. The response of the I-switch is on



timescales of 1–2 min [4], allowing it to be used as a reporter of fine spatial and temporal pH changes associated with biological processes that occur on longer timescales. Many other physiological phenomena such as apoptosis, chemotaxis, viral infection [10], embryogenesis, vesicular recycling and neurodegeneration [16] are known to be modulated by maintenance of pH homeostasis and are compatible with the pH range in which the I-switch is sensitive. Because of its high sensitivity in this pH regime, the I-switch is poised to effectively capture fine changes in pH associated with these pH-correlated phenomena.

It is not at all obvious that functionality *in vitro* should imply functionality *in cellulo* or *in vivo* due to the astounding increase in molecular complexity encountered by the nanodevice as it traverses the *in vitro* to *in cellulo* to *in vivo* boundaries. Thus the quantitative preservation of DNA nanodevice function in complex biological environments is highly encouraging. This opens up possibilities for these systems as test beds for other DNA molecular devices [17] that can be designed to interrogate a variety of processes that track molecularly complex phenomena, thus paving the way for the successful application of DNA nanostructures in biological systems.

**Acknowledgements.** We thank Prof. Satyajit Mayor and Dr. Sandhya P. Koushika for valuable comments and suggestions, Vidhya Rangaraju, Jaffar M. Bhat, Swetha M.G. and Debanjan Goswami for technical input and assistance, Central Imaging Facility at NCBS, the Caenorhabditis Genetics Center (funded by NIH-NCRR) for nematode strains, DBT and the Nanoscience and Technology Initiative of DST for funding. S.S. and S.M. acknowledge the CSIR and YK acknowledges the Innovative Young Biotechnologist Award and Wellcome Trust-DBT India Alliance for fellowships.

## References

1. Shih, W.: Biomolecular self-assembly: dynamic DNA. *Nat. Mater.* 7, 98–100 (2008)
2. Bath, J., Turberfield, A.J.: DNA Nanomachines. *Nat. Nanotech.* 2, 275–284 (2007)
3. Krishnan, Y., Simmel, F.: Nucleic acid based molecular devices. *Angew. Chem. Int. Ed.* 50, 3124–3156 (2011)
4. Modi, S., et al.: A DNA nanomachine that maps spatial and temporal pH changes inside living cells. *Nat. Nanotech.* 4, 325–330 (2009)
5. Guéron, M., Leroy, J.L.: The i-motif in nucleic acids. *Curr. Opin. Struct. Biol.* 10, 326–331 (2000)
6. Guha, A., Sriram, V., Krishnan, K.S., Mayor, S.: Shibire mutations reveal distinct dynamin-independent and -dependent endocytic pathways in primary cultures of *Drosophila* hemocytes. *J. Cell Sci.* 116, 3373–3386 (2003)
7. Fares, H., Greenwald, I.: Genetic analysis of endocytosis in *Caenorhabditis elegans*: Coelomocyte Uptake Defective mutants. *Genetics* 159, 133–145 (2001)
8. Surana, S., Bhat, J.M., Koushika, S.P., Krishnan, Y.: An autonomous DNA nanomachine maps spatiotemporal pH changes in a multicellular living organism. *Nat. Commun.* 2, 340 (2011), doi:10.1038/ncomms1340
9. Overly, C.C., Lee, K.D., Berthiaumet, E., Hollenbeck, P.J.: Quantitative measurement of intraorganelle pH in the endosomal-lysosomal pathway in neurons by using ratiometric imaging with pyranine. *Proc. Natl. Acad. Sci. USA* 92, 3156–3160 (1995)
10. Mukherjee, S., Ghosh, R.N., Maxfield, F.R.: Endocytosis. *Physiol. Rev.* 77, 759–803 (1997)

11. Bucci, C., et al.: The small GTPase rab5 functions as a regulatory factor in the early endocytic pathway. *Cell* 70, 715–728 (1992)
12. Chavrier, P., Parton, R.G., Hauri, H.P., Simons, K., Zerial, M.: Localization of low molecular weight GTP binding proteins to exocytic and endocytic compartments. *Cell* 62, 317–329 (1990)
13. Kostich, M., Fire, A., Fambrough, D.M.: Identification and molecular-genetic characterization of a LAMP/CD68-like protein from *Caenorhabditis elegans*. *J. Cell Sci.* 113, 2595–2606 (2000)
14. Miesenbock, G., De Angelis, D.A., Rothman, J.E.: Visualizing secretion and synaptic transmission with pH-sensitive green fluorescent proteins. *Nature* 394, 192–195 (1998)
15. Ohkuma, S., Poole, B.: Fluorescence probe measurement of the intralysosomal pH in living cells and the perturbation of pH by various agents. *Proc. Natl Acad. Sci. USA* 75, 3327–3331 (1978)
16. Lee, S.-K., Li, W., Ryu, S.-E., Rhim, T.Y., Ahnn, J.: Vacuolar (H<sup>+</sup>)-ATPases in *Caenorhabditis elegans*: What can we learn about giant H<sup>+</sup> pumps from tiny worms? *Biochim. Biophys. Acta* 1797, 1687–1695 (2010)
17. Bhatia, D., Surana, S., Chakraborty, S., Koushika, S.P., Krishnan, Y.: A synthetic icosahedral DNA-based host-cargo complex for functional *in vivo* imaging. *Nat. Commun.* 2, 339 (2011), doi:10.1038/ncomms1337

# Cooperation in an All-RNA Network

Nilesh Vaidya, Jessica Mellor, and Niles Lehman

Department of Chemistry, Portland State University, Portland, OR, USA

The discovery of catalytic RNA molecules (ribozymes) capable of catalyzing a significant number of diverse chemical reactions suggests that an RNA world preceded the DNA-RNA-Protein world we know today. The self-replication of RNA molecules would be the central process of the RNA world. However, a single self-replicating RNA would not sustain information on its own if it surpassed the “error threshold” leading to an error catastrophe (1). On the other hand, a cooperative network of RNA replicators would be able to accumulate, preserve and process information. In the current work, we used a collection of RNA fragments that covalently assemble into an *Azoarcus* group I ribozyme (2) to explore the ability of RNA replicators to form a cooperative catalytic network. Three different constructs of the *Azoarcus* ribozyme with different internal guide sequences (IGS) – GUG (canonical), GAG, and GCG – were designed that are capable of a minimal amount of self-assembly when broken into two fragments. Here, self-assembly depends on a mismatch with non-complementary sequences, CGU, CAU and CUU, respectively, to be recognized by IGS via autocatalysis. Yet when all three constructs are present in the same reaction vessel, concomitant assembly of all three covalent ribozymes is enhanced through an interdependent reaction network. Analysis of these reactions indicates that each system is capable of guiding its own reproduction weakly, along with providing enhanced catalytic support for the reproduction of other constructs through matched and mismatched IGS-tag interactions. The resulting RNA population was observed to evolve over time based on genotyping of more than 50 individual RNAs for various time-points. The unequal catalytic rates of various assembly reactions favor the assembly of one construct over the others. However, the cooperation among all three constructs help all constructs to assemble demonstrating that this RNA network meets many of the requirements of an all RNA hypercycle as envisioned by Eigen (1). Also, when co-incubated with non-interacting (i.e., selfish) yet efficient self-assembly systems, the cooperative assembly outcompetes the selfish self-assembly systems, demonstrating the ability of a cooperative organization to possess an evolutionary advantage (3).

## References

1. Eigen, M., Schuster, P.: The Hypercycle: A principle of natural self-organization. *Die Naturwissenschaften* 64, 541–565 (1977)
2. Hayden, et al.: Systems chemistry on ribozyme self-construction: Evidence for anabolic autocatalysis in a recombination network. *Angew. Chem. Int. Ed.* 47, 8424–8428 (2008)
3. Nowak, M., Highfield, R.: *SuperCooperators: Altruism, Evolution, and Why We Need Each Other to Succeed*. Free Press (2011)

# Designer DNA Architectures for Bionanotechnology

Hao Yan

Department of Chemistry and Biochemistry &  
The Biodesign Institute  
Arizona State University  
Tempe, AZ  
hao.yan@asu.edu

The central task of nanotechnology is to control motions and organize matter with nanometer precision. To achieve this, scientists have investigated a large variety of materials including inorganic materials, organic molecules, and biological polymers as well as different methods that can be sorted into so-called “bottom-up” and “top-down” approaches. Among all of the remarkable achievements made, the success of DNA self-assembly in building programmable nanopatterns has attracted broad attention. Self-assembling DNA nanostructures assembled in this fashion can be modified in a number of ways to contain functional materials with useful biological and electronic properties. This ‘bottom-up’ type of approach has enormous value in the development of “molecular printboards” with resolution exceeding current nanolithographic methods. This talk will discuss some of our recent progress in using DNA as an information-coding polymer for bionanotechnology applications. Specifically, I will discuss our strategy of engineering 3D DNA origami architectures with complex curvatures and discuss perspectives of how to scale up DNA origami nano-constructions. I will use DNA directed self-assembly of hetero-elements as examples to demonstrate potential of structural DNA nanotechnology in practical applications ranging from biophysics to nano-theronostics to energy transfers.

# An Improved DNA-Sticker Addition Algorithm and Its Application to Logarithmic Arithmetic

Mark G. Arnold

Lehigh University, Bethlehem, PA 18015

**Abstract.** The sticker model of computation, implemented using robotic processing of DNA, manipulates in parallel many bitstrings, called strands, that are contained in a limited number of tubes. Prior sticker-addition algorithms are patterned on digital-electronic full-adders that generate carry bits, each of which must be saved in the strand, which involves wasting the strand or using a clear operation (whose biochemical implementation may be problematic). This paper proposes a new sticker-addition algorithm which does not need to record the carry bits. Instead, which tube holds a particular strand implicitly describes whether or not a carry is required. The speed and number of tubes needed are about half that needed by the prior approach. An example is given for real-valued Euclidian norms using the Logarithmic Number System.

**Keywords:** DNA arithmetic, sticker system, addition, Logarithmic Number System (LNS).

## 1 Introduction

Roweis et al. [20] proposed a simple model, called the *sticker system*, for massively-parallel manipulation of  $k$ -bit wide strings (called strands) that are contained in a finite set of containers (called tubes). This model can be implemented using robotic processing of DNA with parallelism possibly approaching the Avogadro constant if the reliability of biochemical processing allows each value to be represented by a single molecule of DNA. Even if redundant molecules are required for reliability, the potential for parallelism exceeds what is currently possible with electronic supercomputers [14]. Despite promising theoretical performance, such DNA supercomputers based on the sticker model have not been realized over the past decade. Beyond the obvious technological challenges of robotic manipulation of DNA in solution, is the perception that the sticker model is so simple as to be cumbersome and inefficient for applications that bear any resemblance to those on actual supercomputers, in which arithmetic often dominates. This paper offers significant improvement to sticker arithmetic, which might make numeric supercomputer-like code more feasible with the sticker model, and thereby encourage implementors to pursue further developments of sticker hardware.

In the sticker system, each strand consists of a long single-stranded DNA molecule, together with “stickers” (short complementary DNA molecules designed to hybridize at only a particular position along the long strand). The distinction between the bit ‘0’ and the bit ‘1’ for an arbitrary bit position along

a particular strand is whether a sticker is present. Since, in theory, that sticker can only hybridize at one place on the strand, introduction of enough sticker molecules that represent a particular bit will cause that bit to become ‘1’ in every strand. Alternatively, a probe molecule, which can later be filtered out from a tube, but that also includes the same complementary DNA bases as a particular sticker allows strands to be separated into two different tubes, depending on whether or not that specific sticker is present. (The probe cannot stick if there is already a sticker at the specific location of interest; if the sticker is not present, the strand will be filtered out with the probe at the position of interest; it is assumed the probes can be melted off and filtered away without disturbing the normal stickers, thereby returning the selected strands to their original format.) Retesting the same bit (“refinery” model) improves reliability[20]. Further including the obvious operation of combining (pouring) the contents of tubes together gives three fundamental operations for the sticker model which have realistic  $O(1)$  biochemical implementations:

- **set**( $t, i$ ): Make bit position  $i$  equal to 1 for every strand of tube  $t$ .
- **separate**( $t_1, t_0, t, i$ ) strands of tube  $t$  into two tubes ( $t_1$  and  $t_0$ ) based on a particular bit position,  $i$ . The order  $t_1, t_0$  is like that of an **if else**.
- **combine**( $t_1, t_0$ ): Pour tube  $t_0$  into the tube  $t_1$ .

Another operation, **clear**( $t, i$ ), which removes the sticker from a specified position in every strand (making the bit position,  $i$ , equal to 0) is normally included among the primitives as it makes conventional Boolean logic much easier; however, its biochemical implementation is perhaps problematic [13]. Omitting the **clear** operation still allows for complicated feedforward computations; however, all intermediate results must be written into a fresh bit. The more intermediate results required, the longer the strand must be. Sticker algorithms with fewer intermediate result bits are therefore desirable. To see the cost influence of number of tubes,  $n_t$ , and number of bits,  $k$ , consider a rudimentary model for the total mass,  $m_t$ . A particular tube might contain all the stickers, which means there is a minimum mass of water per bit needed to act as a solvent in every tube—even if some of the time certain tubes contain no stickers. Let’s say water is  $5m_k$  per bit per tube, where  $m_k$  is the mass per bit of nucleic acids for all strands and stickers. Assume the mass of the MEMS microrobotic components [8,7] is similar to the water it must hold, giving

$$m_t = k(n_t(5m_k + 5m_k) + m_k) = m_k k(10n_t + 1) \approx 10n_t m_k k. \quad (1)$$

The algorithms for the DNA sticker system originally focused on non-numeric applications (typically graph problems with NP-complete solutions on conventional computers) [20,13,22]. In contrast to more sophisticated models of DNA computing [1], the sticker model does not need enzymes, and, in theory, allows the constituent molecules to be recycled. In contrast to more recent proposals for autonomous DNA- and molecular-based computation (i.e., digital-circuit cascades using strand displacement [19]), the sticker model requires conventional computers (or digital-logic controllers) to sequence the robotic biochemical operations that implement each step [5]. As such, the sticker model is a candidate for a

kind of (as yet unachieved) supercomputer architecture. One could view a sticker computer and a Graphics Processing Unit (GPU) as being quite analogous: each sticker is like a thread—*independent data being processed concurrently by one (kernel) program*. Both stickers and GPU hardware need to be attached to conventional computers to be useful. The distinction is that the sticker computer has only these slow, very elementary, single-bit biochemical operations compared to the fast 32-bit electronic floating-point operations of a GPU. What makes up for this is that there are massively more strands possible with stickers than there are threads possible on the largest GPU supercomputers.

There are three aspects where one could make a distinction between sequential and parallel processing: at the bit level, at the tube level and at the strand level. In the sticker model, bit processing is always sequential; strand processing is always in parallel; tube processing might be either way. Just like electronic microprocessors may or may not have parallel datapath elements (e.g., busses) that can process independent variables in parallel, the sticker model may or may not have parallel tubes that can process independent sets of strands in parallel. In other words, at a given moment in time, distinct tubes might have different commands (like **set** and **separate**) occurring at the same instant, provided the digital controller can issue simultaneous commands, and the robotic system can execute them simultaneously. This paper will consider both tube-serial and tube-parallel versions of the sticker method.

Many early proposals for DNA-based arithmetic used techniques more involved than the sticker system. Guarnieri et al. [10,11] proposed a one-pot approach for adding two non-negative integers, with the limitation that the input format is different than the output format. Yurke et al. [24] described a technique in which input integers are given on separate strands. Barua [4] showed how the integers could have the same input and output DNA structure. LaBean et al. [16] describe self-organized weaving of three-helix DNA complexes to create a sum, which in a complicated computation leaves a record of intermediate results. Most authors have considered integer (rather than real) arithmetic; an exception is de Santis et al. [9], who describes representing one very-high-precision floating-point number with DNA strands. Section 6 of this paper has the opposite goal: representing a huge (Avogadro-sized) set of low-precision real numbers.

Recently, there has been interest in developing computer-arithmetic algorithms [23,12] using the simple tube-parallel sticker system. Having operations such as addition, multiplication and division available on a sticker-based computer opens up a larger set of possible applications, for example, Chang et al. [6] use sticker-based subtraction and division similar to [23,12] in cryptanalysis of RSA public-key ciphers.

Addition is the foundation of most arithmetic algorithms, and in binary, integer addition itself is normally built from 1-bit wide adders (so called “full adders”) that have three inputs (two data bits and a carry input). Section 2 reviews sticker-addition algorithms in the sticker literature [23,12,6] that follow prior-art practices for digital-electronic full-adder design that generate carry bits for each bit position in the input. With the sticker method, this approach either

requires that an intermediary bit be recorded in the strand for every input bit [6] or that a single carry bit must be set and cleared many times during the bit-sequential processing required to add two numbers [23,12]. Because the biochemical basis for the `clear` operation is problematic, algorithms that reuse a single carry bit will be more difficult to implement in the lab; yet it becomes quite cumbersome to record every carry for algorithms that need multiple additions (even algorithms as simple as  $n$ -bit multiplication need  $O(n^2)$  carries).

This paper proposes a new sticker-addition algorithm, given in Section 3, which does not need to record the carry bits. Instead, the carry bit is implicitly described by which tube a particular strand is in at each stage of the algorithm. With this novel approach, the speed (number of biochemical steps) and cost (number of tubes needed) of sticker-based addition are improved significantly compared to prior approaches. Section 4 gives a short example to illustrate the operation of the new algorithms with two-bit inputs. Section 5 considers a simplification possible when adding a constant. Because the proposed algorithms produce only the final sum bits and no intermediate results, more extensive algorithms can fit into reasonable-sized strands without needing the `clear` operation. Section 6 gives a brief overview of such an algorithm, where the novel integer-valued addition algorithm proposed here improves a sticker computation for the real-valued Euclidian norm using the Logarithmic Number System [21], which transforms difficult operations, like multiplication, to now much simplified integer addition. Section 7 presents conclusions.

## 2 Prior Algorithms

In electronic digital design, a gate with an  $n$ -bit input needs a truth table with  $2^n$  entries. In the sticker system, this can be implemented by separating strands into  $2^n$  tubes based on these  $n$  bits. A naïve way to translate conventional digital designs into the sticker system is to realize the truth tables for common two-input gates with sticker code, and synthesize larger gates from such two-input primitives. In the following code<sup>1</sup>:

```
function xor(t,o,i,j)
{
  separate(t+1, t, t, i);
  separate(t+3, t+1, t+1, j);
  separate(t+2, t, t, j);
  combine(t+1,t+2);
  set(t+1,o);
  combine(t,t+3);
  clear(t,o);
  combine(t,t+1);
}
```

---

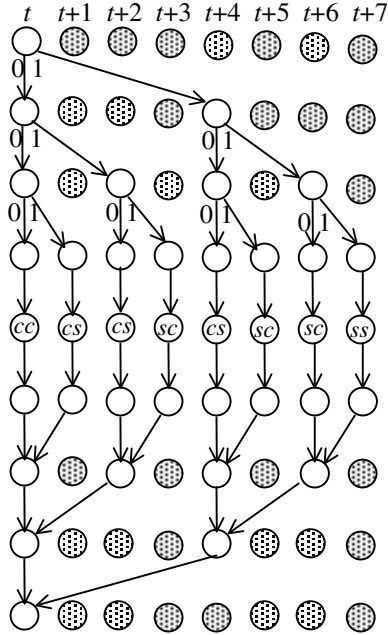
<sup>1</sup> The syntax given is compatible with the JavaScript simulator at [www.xlnsresearch.com/sticker.htm](http://www.xlnsresearch.com/sticker.htm) except that the `separate()` operation used in this paper is called `separate3()` in the simulator.



arguments  $t$ ,  $o$ ,  $i$  and  $j$  on the sticker system's digital-electronic controller allow us to describe the algorithm in a generic-enough way that it may be reused in many applications. Here we assume `clear` is permitted. This two-input exclusive-OR needs four tubes (numbered  $t$  through  $t+3$ ). Let  $o$  be the bit position within the strand of the output bit, and  $i$  and  $j$  be the bit positions of the two input bits. Assume tube  $t$  contains all input strands at the beginning and all output strands at the end. After the `separate` operations, tube numbers  $t$  through  $t+3$  (often called  $T_{00}$  through  $T_{11}$  in the literature [12]) correspond to the individual lines of the exclusive-OR truth table. In theory, each tube needs a `set` or `clear`, depending on the output listed in the truth table. Combining all tubes whose output bit is 1 and similarly all tubes whose output bit is 0 means the equivalent may be accomplished with a single `set` and a single `clear`. Combining those two tubes together forms a single tube (numbered  $t$ ) of all output strands. Only four tubes are required for an arbitrarily complex algorithm; the disadvantage is that it is very slow (eight steps which take seven units of time if tube parallelism is allowed for the last two `separates`). Similar code allows implementation of other two-input gates, like `and`. Combining these and reusing two intermediate result bits ( $c-1$  and  $c$ ) many times allows us to create a full adder with four tubes that requires about eighty time-steps:

```
function fulladdgate(t,c,o,i,j)
{
  xor(t, c, i, j);
  xor(t, o, c, c-1);
  and(t, c-1, c, c-1);
  and(t, c, i, j);
  xor(t, c-1, c-1, c);
}
```

Tube-steps (number-of-tubes multiplied by time-steps) are a measure of algorithm cost. In this case, we need  $4 \cdot 80 = 320$  tube-steps. The inefficiency of this approach is why [23,12,6] have increased the number of tubes used in exchange for significant increase in speed. In particular, instead of conceptualizing the problem as being composed of two-input gates, the prior sticker adders have implemented the full-adder as a 3-input function requiring  $2^3$  tubes. Fig. 1 shows how the eight tubes are used by the prior algorithms ([23,12,6] assume tube parallelism) for nine time steps. Fig. 1 shows each tube as a circle: the columns represent distinct tubes; the rows represent distinct times. A shaded circle is a tube that is unused at the time shown; an unshaded circle with two labeled arrows coming out is a `separate`; an unshaded circle with two unlabeled arrows coming in is a `combine`. The `set` and `clear` are described inside each circle as  $cc$ ,  $cs$ ,  $sc$  and  $ss$  corresponding to the four possible patterns of bits required, where the left bit is the carry and the right bit is the sum, e.g.,  $sc$  sets the carry and clears the sum. Fig. 1 needs 72 tube steps, of which 34 are unused, i.e., 53 percent utilization. Although the algorithm of [23,12,6] is a significant improvement over the 320 time steps required by the naïve approach, it does come with the drawback of doubling the number of tubes (which may be significant in



**Fig. 1.** Prior addition algorithm [12] needs eight tubes and uses `clear`

algorithms that perform additions in distinct tubes concurrently, as illustrated in Section 6). The algorithm proposed in the next section does not use as many tubes as [23,12,6], cuts the number of time steps to less than one half, does not use `clear`, and does not need intermediate results.

### 3 Novel Algorithm

Instead of using tube number  $\tau$  as the only input tube for the algorithm, the novel idea in this paper is to define the input to the full adder as coming from two tubes that contain distinct strands. Tube  $\tau$ , similar to the earlier examples, contains strands for which there is not currently a carry required; tube  $\tau+2$  contains different strands where previous processing indicates a carry is required. Another way of looking this is to say: if a particular strand were input via tube  $\tau+2$ , the resulting sum would be one more than if the same strand had been input via tube  $\tau$ . Likewise, this algorithm uses both tubes  $\tau$  and  $\tau+2$  as outputs: the output in tube  $\tau$  is correct as is (for as many bits as have been processed); the output in tube  $\tau+2$  needs to have a carry added to its next most significant place. In the novel algorithm described below, the Greek-name comments and the associated Greek letters in Fig. 2 a) help clarify the example in the next section.

```

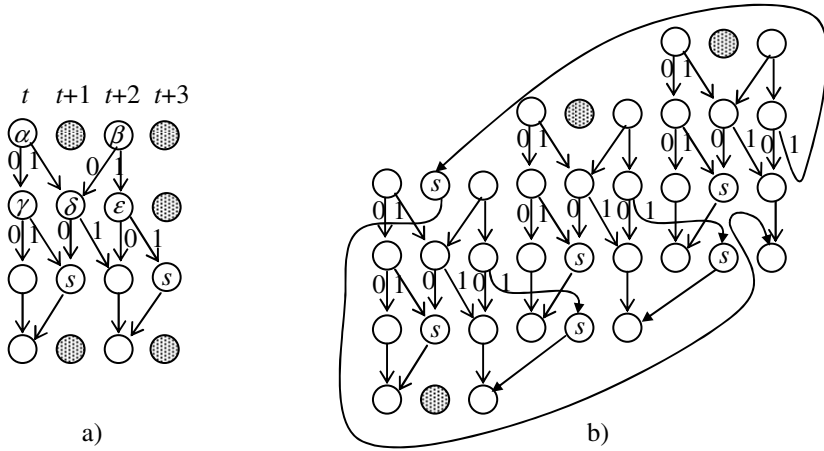
function fulladd(t,o,i,j)
{
  separate(t+1,t, t, i); //alpha
  separate(t+2,t+1,t+2,i); //beta
  separate(t+3,t+2,t+2,j); //gamma
  separate(t+2,t+1,t+1,j); //delta
  separate(t+1,t ,t ,j); //epsilon
  set(t+1,o);
  set(t+3,o);
  combine(t, t+1);
  combine(t+2,t+3);
}

```

Because of dependencies, the order of the last three ( $\gamma$ ,  $\delta$  and  $\epsilon$ ) `separates` needs to be as shown if this algorithm is carried out on a tube-sequential machine; the effect on a tube-parallel machine works the same regardless of how these are arranged. (Optimized compilation for parallel machines in light of dependences [15] is a long-studied issue in computer science; the novel algorithm here is simple enough to be optimized by hand.) As shown in Fig. 2 a), the novel algorithm needs only four tubes during four time steps (16 tube-steps) with only five tube steps unused (69 percent utilization). Fig. 2 has one additional notation: a circle with two labeled arrows coming in is the destination of multiple `separates` from the previous time—the tube-parallel model [20,5] assumes enough transducer/pump hardware is available to allow this. Because Fig. 2 a) only needs four tubes, the total number of tubes in an algorithm that performs concurrent additions in distinct sets of tubes will be half of that needed by the prior algorithm [23,12,6]. Furthermore, since tube `t+3` is used only once, reallocating three unused tube-steps and interleaving three distinct sets of tubes, as shown in Fig. 2 b), allows three sets of values to be added concurrently using only nine tubes (three tubes, 12 tube-steps, and 92 percent utilization per sum, a significant improvement over [23,12,6]). Section 6 describes an application that benefits from this.

## 4 Example

Performing an  $n$ -bit addition requires invoking `fulladd`  $n$  times on successive input and output bit positions. If only a single addition is needed, `t` may be fixed at 0, meaning the algorithm uses tubes 0 through 3. With  $n = 2$ , each input number ranges between 0 to  $2^n - 1 = 3$ . These inputs are concatenated together, along with an  $(n + 1)$ -bit field containing zeros that will be replaced with the result of the addition (because no stickers are here at the start, `clear` is not needed). In other words, the total length of the strand is  $3n + 1$ . We use underscore to distinguish between these three components of each strand. Like [20,13,22], we start in tube number 0 with the set of all ( $2^{2n} = 16$ ) possible input combinations:



**Fig. 2.** New add is faster, and avoids clear: a) 4 tubes/sum, b) interleaved 3 tubes/sum

```
{0_0_0,0_0_1,0_0_2,0_0_3,0_1_0,0_1_1,0_1_2,0_1_3,
 0_2_0,0_2_1,0_2_2,0_2_3,0_3_0,0_3_1,0_3_2,0_3_3}
```

Each left-most zero will become the sum of the adjacent data in each strand. The code and Fig. 2 a) are re-used for all iterations; tube 2 is empty only during the first (least-significant) iteration since we start with no carries. The first ( $\alpha$ ) **separate** removes the strands with odd values in the first (right-most) input from tube 0, leaving the even values ( $_0$  and  $_2$ ) in tube 0:

```
{0_0_0,0_0_2,0_1_0,0_1_2,0_2_0,0_2_2,0_3_0,0_3_2}
```

This **separate** also puts the strands with odd values ( $_1$  and  $_3$ ) into tube 1:

```
{0_0_1,0_0_3,0_1_1,0_1_3,0_2_1,0_2_3,0_3_1,0_3_3}
```

The second ( $\beta$ ) and third ( $\gamma$ ) **separates** do nothing on this iteration since tube 2 is empty. The fourth ( $\delta$ ) **separate** further subdivides the strands in tube 1: it keeps the strands with even second inputs ( $_0_$  or  $_2_$ ) in tube 1:

```
{0_0_1,0_0_3,0_2_1,0_2_3}
```

and puts the strands with odd values ( $_1_$  or  $_3_$ ) for this second input in tube 2:

```
{0_1_1,0_1_3,0_3_1,0_3_3}
```

In other words, at this point, tube 1 contains strands with inputs whose sum will be odd; tube 2 contains strands with inputs whose sum will be even. The fifth ( $\epsilon$ ) **separate** transfers the strands from tube 0 whose second input is odd to tube 1 (which already had other strands from a previous operation). The effect of this is that the strands in tube 1 have inputs whose sum will be odd:

```
{0_0_1,0_0_3,0_1_0,0_1_2,0_2_1,0_2_3,0_3_0,0_3_2}
```

which leaves tube 0 to hold the strands that have inputs whose sum will be even:

$$\{0_0_0, 0_0_2, 0_2_0, 0_2_2\}$$

At this stage, tubes 0 and 2 have the correct low-order sum bit (0), and so nothing has to be done with them. Tube 1 needs to have this low-order bit **set**, giving:

$$\{1_0_1, 1_0_3, 1_1_0, 1_1_2, 1_2_1, 1_2_3, 1_3_0, 1_3_2\}$$

The other **set** (for tube 3) does nothing in this case since tube 3 is empty. The first **combine** operation merges tubes 0 and 1, producing in tube 0 the set of strands that do not cause a carry into the next place:

$$\{0_0_0, 1_0_1, 0_0_2, 1_0_3, 1_1_0, 1_1_2, 0_2_0, 1_2_1, 0_2_2, 1_2_3, 1_3_0, 1_3_2\}$$

The final **combine** (of the first iteration) merges tube 2 with the empty tube 3, producing in tube 2 the set of strands for which there will be a carry into the next place:

$$\{0_1_1, 0_1_3, 0_3_1, 0_3_3\}$$

Because tube 2 is now not empty, the second iteration will be somewhat more involved.

The first ( $\alpha$ ) **separate** of the second iteration looks at the most-significant bit (of the two bits) for the first input in tube 0 (the one assuming no carries): it puts strands where this bit is 0 (where the first input is  $< 2$ ) into tube 0:

$$\{0_0_0, 1_0_1, 1_1_0, 0_2_0, 1_2_1, 1_3_0\}$$

and puts strands where this bit is 1 (where the first input is  $\geq 2$ ) into tube 1:

$$\{0_0_2, 1_0_3, 1_1_2, 0_2_2, 1_2_3, 1_3_2\}$$

The second ( $\beta$ ) **separate** of this iteration also looks at the most-significant bit for the first input, but in this case for the data in tube 2. This **separate** puts strands where this bit is zero into tube 1 (merging with the data already there):

$$\{0_0_2, 1_0_3, 0_1_1, 1_1_2, 0_2_2, 1_2_3, 0_3_1, 1_3_2\}$$

and leaves strands where this bit is 1 in tube 2:

$$\{0_1_3, 0_3_3\}$$

The third ( $\gamma$ ) **separate** of this iteration looks at the most-significant bit for the second input in tube 2, and puts strands where this bit is zero into tube 2:

$$\{0_1_3\}$$

Also, it puts strands where this bit is 1 into tube 3:

$$\{0_3_3\}$$

The fourth ( $\delta$ ) **separate** looks at the most-significant bit for the second input in tube 1, and puts strands where this most-significant bit is 1 into tube 2 (along with the strands already in tube 2):

{0\_1\_3,0\_2\_2,1\_2\_3,0\_3\_1,1\_3\_2}

leaving the other strands in tube 1:

{0\_0\_2,1\_0\_3,0\_1\_1,1\_1\_2}

The final ( $\epsilon$ ) **separate** looks at the most-significant bit for the second input in tube 0, and puts strands where this most-significant bit is 1 into tube 1 (along with the strands already in tube 1):

{0\_0\_2,1\_0\_3,0\_1\_1,1\_1\_2,0\_2\_0,1\_2\_1,1\_3\_0}

leaving the others in tube 0:

{0\_0\_0,1\_0\_1,1\_1\_0}

Next, we **set** (in this example, the most-significant) bit in the output of tube 1:

{2\_0\_2,3\_0\_3,2\_1\_1,3\_1\_2,2\_2\_0,3\_2\_1,3\_3\_0}

Also, we **set** the most-significant bit in the output of tube 3:

{2\_3\_3}

In the next-to-last step of the code, tubes 0 and 1 are combined into tube 0:

{0\_0\_0,1\_0\_1,2\_0\_2,3\_0\_3,1\_1\_0,2\_1\_1,3\_1\_2,2\_2\_0,3\_2\_1,3\_3\_0}

which is the correct set of sums (which would need no carry even if there were an additional place). Finally, tubes 2 and 3 are combined into tube 2:

{0\_1\_3,0\_2\_2,1\_2\_3,0\_3\_1,1\_3\_2,2\_3\_3}

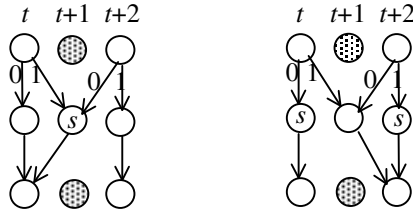
which is the correct set of modulo-4 sums (which would need a carry if there were an additional place) Since this is final bit of the example, outside the iteration listed in the code, a final **set** of bit 2 in tube 2 yields the correct  $n + 1$ -bit sums:

{4\_1\_3,4\_2\_2,5\_2\_3,4\_3\_1,5\_3\_2,6\_3\_3}

Tubes 0 and 2 could be combined if required, or could be kept distinct (useful if one of the inputs represents a two's-complement value for a comparison). If executed sequentially, the proposed  $n = 2$  example requires 20 time steps; if executed in tube-parallel fashion, this example requires  $4n + 2 = 8$  time steps (the extra two steps occurring outside the main iteration to complete a single set of  $(n + 1)$ -bit outputs—other applications, like modulo- $2^n$  addition or the comparison mentioned above, might require less). In contrast to our novel method, Guo et al. [12] require  $9n = 18$  time steps for tube-parallel addition, with all of the other disadvantages mentioned previously.

## 5 Adding Constants

There are situations when a constant binary number needs to be added to a value encoded on the strands. Although one could use **set** and **clear** to place



**Fig. 3.** Half adders: (a) assuming constant bit is 0; (b) assuming constant bit is 1

the constant on all of the strands before invoking the above algorithm, this would waste part of the strand. Instead, it would be better to compose the addition of the constant out of a sequence of half adders. The number to which the constant is added is represented by which tube the strand is in, as in the previous section. There are two kinds of such half adders: one for when a particular bit of the constant is 0, the other for when that bit of the constant is 1. Fig. 3 shows both types of half adders, which both use three tubes and three time steps.

## 6 Logarithmic Number System

The Logarithmic Number System (LNS) [21] represents a real value (denoted by upper-case  $X$ ) with its base- $b$  logarithm encoded as an  $n$ -bit fixed-point number  $x = \text{int}(2^f \cdot \log_b(X))/2^f$ , where the high  $n - f$  bits provide dynamic range and the remaining  $f$  bits provide the precision. Typically, designers choose  $b = 2$ , which makes LNS similar to Floating Point (FP). This paper will use the term Floating-Point Operation Per Second (FLOPS) even though the operations are being implemented by LNS. LNS has been most useful in applications where modest  $n$  is acceptable, for example the GRAVity PipelinE (GRAPE) supercomputers [17] (which won the Gordon-Bell Prize) use  $n = 13$  LNS to accelerate  $N$ -body calculations. In LNS, powers, multiplication and division are very easy, involving only integer shifting, addition and subtraction, which can be implemented in  $O(n)$  time with the sticker system without intermediate results because of the novel integer addition algorithm of Fig. 2. In contrast, conventional multiplication (even with the improved addition algorithm proposed in this paper) requires  $O(n^2)$  time and intermediate bits for partial products.

Mitchell [18] suggested a very low-cost approach to approximating the base-2 logarithm and antilogarithm without needing a table or iteration. Of interest here is Mitchell's antilogarithm, which is based on the observation  $2^x \approx x + 1$  in the range  $0 \leq x \leq 1$ . This can be generalized as  $2^x \approx 2^{\text{int}(x)} \cdot (\text{frac}(x) + 1)$ .

As an example, suppose we would like to compute the Euclidian norm,  $R = \sqrt{X^2 + Y^2}$ , of the real  $(X, Y)$ , which is a simplified example similar to the gravitational force calculation at the heart of GRAPE. One approach for the norm would be: take the logarithms of the numbers; shift them left (i.e., multiply each of them by two); take the antilogarithms; add them in conventional fixed-point;

take the logarithm of the sum; shift it right (i.e., divide by two); and take the antilogarithm. This is cumbersome, and would require many intermediate bits in the sticker system. In order to take advantage of LNS, it is better to keep numbers in logarithmic format throughout the entire computation. The point then is how addition operates if it is not done in fixed point. Given  $x \approx \log_b(X)$  and  $y \approx \log_b(Y)$ , we can compute the logarithm of the sum with  $\log_b(X+Y) = y + s_b(x-y)$ , where  $s_b(z) = \log_b(1+b^z)$  [21,17,3]. This function has the following properties:  $s_b(z) \approx 0$  for  $z < -f$  and  $s_b(z) = s_b(-z) + z$ . The latter means  $\log_b(X+Y) = \max(x, y) + s_b(-|x-y|)$ ; the former means  $\log_b(X+Y) = \max(x, y)$  for  $|x-y| \geq f$ . The norm may be computed as  $r = \log_b(R) = \max(x, y) + s_b(-2|x-y|)/2$ . (This paper considers only addition of positive reals, which is sufficient for the norm; a sign bit [21] would allow negative reals and subtraction at about double the cost.) For  $b = 2$ , it has been suggested to use parallel [2] or serial [3] digital-electronic implementations of Mitchell's method as an approximation for the LNS addition function:  $s_2(z) \approx 2^z \approx 2^{\text{int}(z)} \cdot (\text{frac}(z) + 1)$  for  $-f < z < 0$ ; however, to the best of the author's knowledge, LNS has never been used with the sticker system.

Such LNS addition can be implemented with a novel sticker approach using two phases. First, compute

$$z = x - y \tag{2}$$

using four tubes with a full subtractor analogous to the full adder in Fig. 2 a). This  $z$  is the only intermediate result required to be written into the strands. Second, separate the strands into  $2f$  different sets of tubes based on the bits of  $\text{int}(z)$ . Half of them deal with positive  $z$  and half with negative  $z$ . Each set of tubes consists of three tubes, as required in Fig. 2 b). Included are two sets of tubes to deal with extremely large ( $z \geq f - 1$ ) or small ( $z \leq -f$ ) cases. These large and small cases are separated based on the high-order bits of  $\text{int}(z)$ . The remaining cases are separated based on the low-order bits of  $\text{int}(z)$ . For example, if  $f = 8$ , there are eight cases for negative  $z$ :

$$t_{45} = y + 2^{-1} + 2^{-1} \cdot \text{frac}(z), \text{ if } \text{int}(z) = -1 \tag{3}$$

$$t_{42} = y + 2^{-2} + 2^{-2} \cdot \text{frac}(z), \text{ if } \text{int}(z) = -2 \tag{4}$$

$$t_{39} = y + 2^{-3} + 2^{-3} \cdot \text{frac}(z), \text{ if } \text{int}(z) = -3 \tag{5}$$

$$t_{36} = y + 2^{-4} + 2^{-4} \cdot \text{frac}(z), \text{ if } \text{int}(z) = -4 \tag{6}$$

$$t_{33} = y + 2^{-5} + 2^{-5} \cdot \text{frac}(z), \text{ if } \text{int}(z) = -5 \tag{7}$$

$$t_{30} = y + 2^{-6} + 2^{-6} \cdot \text{frac}(z), \text{ if } \text{int}(z) = -6 \tag{8}$$

$$t_{27} = y + 2^{-7} + 2^{-7} \cdot \text{frac}(z), \text{ if } \text{int}(z) = -7 \tag{9}$$

$$t_{24} = y, \text{ if } \text{int}(z) \leq -8 \tag{10}$$

where  $t_{45}$ ,  $t_{42}$  etc. actually need three tubes to compute the associated additions using interleaving. Mathematically, the power-of-two constant and the shifted  $z$  bits occupy distinct positions. This means the process starts using  $f + \text{int}(z)$  invocations of `fulladd`; then it uses `halfadd1` once (for the power-of-two constant);





**Fig. 4.** Norm  $R = \sqrt{X^2 + Y^2}$  computed by: (2)–(18) LNS (left); FP (right)

finally, finishing the process with several invocations of `halfadd0` (to deal with the possibility of a succession of carries in what otherwise would be a zero result). Because (3)–(10) are defined for negative  $z$ , we need to transform for the cases of positive  $z$ :

$$t_0 = x + 2^0 - 2^{-1} \cdot \text{frac}(z), \quad \mathbf{if} \text{ int}(z) = 0 \quad (11)$$

$$t_3 = x + 2^{-1} - 2^{-2} \cdot \text{frac}(z), \quad \mathbf{if} \text{ int}(z) = 1 \quad (12)$$

$$t_6 = x + 2^{-2} - 2^{-3} \cdot \text{frac}(z), \quad \mathbf{if} \text{ int}(z) = 2 \quad (13)$$

$$t_9 = x + 2^{-3} - 2^{-4} \cdot \text{frac}(z), \quad \mathbf{if} \text{ int}(z) = 3 \quad (14)$$

$$t_{12} = x + 2^{-4} - 2^{-5} \cdot \text{frac}(z), \quad \mathbf{if} \text{ int}(z) = 4 \quad (15)$$

$$t_{15} = x + 2^{-5} - 2^{-6} \cdot \text{frac}(z), \quad \mathbf{if} \text{ int}(z) = 5 \quad (16)$$

$$t_{18} = x + 2^{-6} - 2^{-7} \cdot \text{frac}(z), \quad \mathbf{if} \text{ int}(z) = 6 \quad (17)$$

$$t_{21} = x, \quad \mathbf{if} \text{ int}(z) \geq 7 \quad (18)$$

There are sixteen distinct sets of strands, being processed in parallel within 48 tubes. The proposed integer addition algorithm makes significant savings. It eliminates 80 tubes compared to what would be needed if the Guo et al. adder [12] were used. It also cuts the execution time roughly in half.

Since square and square root are implemented in LNS with trivial shifting (i.e., which index that is passed from the digital controller to the `fulladd`, etc. routines), this is all that is required to implement the norm (conventionally considered as four operations: two squares, one add and one square root). The time for the norm will be around  $8n$ , or  $2n$  per conventional operation. Fig. 4 shows norms computed via the novel LNS (2)–(18) and conventional FP approaches seem identical, although the shorter LNS representation inherently has greater relative error not visible at this scale. ([18,2,3] give error analyses.) This is acceptable in some scientific simulations, like GRAPE, because fast algorithms (e.g., Barnes-Hut) often make approximations that overwhelm any arithmetic issues of this scale [17]. Conventionally, sticker computation has used the set of all possible input combinations [20] or some subset [13] chosen at random. The inclusion of LNS capabilities allow such uniform inputs to be reshaped according

to real-valued distributions, for example, instead of starting with a uniform grid of stars in an  $N$ -body simulation, realistic-looking galaxies could be formed first using formulae similar to the example.

Let's say  $n = 16$  (bigger than GRAPE) and we have an ideal sticker supercomputer that can process  $6 \cdot 10^{23}$  strands in a second (with microfluidic droplets, rather than a macroscopic test tube, reaction rates may increase [7] over previous assumptions [20]), making 10 zettaFLOPS, i.e.,  $10^4$  faster than the exaFLOPS electronic supercomputers predicted by decade's end. [14] The nucleic acids' mass per bit,  $m_k$ , would be about one kg. From (1), the proposed LNS  $n_t = 48$ -tube,  $k = 64$ -bit,  $m_t = 3 \cdot 10^4$ -kg DNA supercomputer might be  $10^3$  lighter than the "warehouse full of computers" [14] needed for conventional-exaFLOPS. Of course, if a sticker supercomputer is ever realized, it is unlikely to achieve ideal performance, but a speed-mass factor of  $10^7$  gives some margin within the next decade for DNA supercomputers to outperform conventional supercomputers for specialized LNS-based numerical applications, like GRAPE, despite what may be overly optimistic estimates here. For comparison, the same LNS (2)–(18) norm with a `clear-less-Guo` adder would be half the speed and require  $n_t = 128$ ,  $k = 96$ ,  $m_t = 1.2 \cdot 10^5$ —almost an order of magnitude inferior speed-mass. An  $n = 16$ ,  $f = 8$  FP norm with a `clear-less-Guo` adder needs similar  $n_t$  (for FP add) but  $k \approx 10^3$  (for intermediate FP bits) with  $n$ -fold slowdown—nearly two additional orders of magnitude worse speed-mass.

## 7 Conclusions

This paper described a new integer addition algorithm for the sticker model of DNA computation. The proposed algorithm does not need to record the carry bits, which otherwise would have to be saved in the strand (either wasting space for intermediate bits or needing a `clear` operation with problematic biochemical implementation). Also, the proposed algorithm uses half the time and tubes compared to prior approaches because it implicitly describes the carry by which tube a particular strand is in. Further improvements are suggested for adding constants and interleaving concurrent additions. The proposed integer addition algorithm could improve any DNA-sticker arithmetic method (such as [6]), but algorithms like  $n$ -bit integer multiplication will still need  $O(n^2)$  intermediate bits. In applications working with approximate real values, transforming to logarithmic representation allows multiplication, division and square root to be substituted with operations that involve no intermediate bits. Using this LNS approach (where different sums need to be computed in parallel by different sets of tubes), an example real-valued Euclidian norm capitalizes on the cost savings of the proposed integer addition. A back-of-the-envelope estimate indicates more than a million-fold speed-mass advantage compared to conventional supercomputing. Of this, perhaps two or three orders of magnitude are from sticker-size, tube and speed advantages of the proposed integer-addition and LNS algorithms.

## References

1. Adleman, L.: Molecular Computation of Solutions to Combinatorial Problems. *Science* 266, 1021–1024 (1994)
2. Arnold, M.G.: LPVIP: A Low-Power ROM-Less ALU for Low-Precision LNS. In: Macii, E., Paliouras, V., Koufopavlou, O. (eds.) *PATMOS 2004*. LNCS, vol. 3254, pp. 675–684. Springer, Heidelberg (2004)
3. Arnold, M.G., Vouziz, P.: A Serial Logarithmic Number System ALU. In: Kubatova, H. (ed.) *2007 EuroMicro DSD*, pp. 151–154. IEEE Press, Los Alamitos (2007)
4. Barua, R.: Binary Arithmetic for DNA Computer. *DNA Comp.* 8, 124–132 (2002)
5. Carroll, S.: A Complete Programming Environment for DNA Computation. In: *Workshop Non-Silicon Comp. NSC-1*, pp. 46–53 (2002)
6. Chang, W.-L., et al.: Fast Parallel Molecular Algorithms for DNA-Based Computation: Factoring Integers. *IEEE Trans. Nanobiosci.* 4, 149–163 (2005)
7. Chakrabarthi, K., et al.: Design Tools for Digital Microfluidic Biochips. *IEEE Trans. Comp.-Aid. Des* 29, 1001–1017 (2010)
8. Cho, J.H., et al.: Reconfigurable Multi-Component Sensors Built from MEMS Payloads Carried by Micro-Robots. In: *Sensor App. Symp.*, pp. 15–19 (2010)
9. de Santis, F., et al.: A DNA ALU. *WSEAS Trans. Bio. Biomed.* 1, 436–440 (2004)
10. Guarnieri, F., et al.: Making DNA Add. *Science* 273, 220–223 (1996)
11. Guarnieri, F., Bancroft, C.: Use of a Horizontal Chain Reaction for DNA-based Addition. *DIMACS* 44, 105–111 (1999)
12. Guo, P., Zhang, H.: DNA Implementation of Arithmetic Operations. In: *2009 Int. Conf. Natural Comp.*, pp. 153–159 (2009)
13. Ignatova, Z., Martinez-Perez, I., Zimmermann, K.: DNA Computing Models, Section 5.3. Springer, New York (2008)
14. Kogge, P.: Next-Generation Supercomputers. *IEEE Spectrum* 48(2), 48–54 (2011)
15. Kuck, D.: *Structure of Computers and Computations*. Wiley, New York (1978)
16. LaBean, T.H., Winfree, E., Reif, J.H.: Experimental Progress in Computation by Self-Assembly of DNA Tilings. In: *DNA Based Computers*, pp. 123–140 (1999)
17. Makino, J., Taiji, M.: *Scientific Simulations with Special-Purpose Computers—the GRAPE Systems*. Wiley, Chichester (1998)
18. Mitchell, J.N.: Computer Multiplication and Division using Binary Logarithms. *IEEE Trans. Elec. Computers EC-11*, 512–517 (1962)
19. Qian, L., Winfree, E.: A Simple DNA Gate Motif for Synthesizing Large-scale Circuits. *J. R. Soc. Interface* (2011), doi:10.1098/rsif.2010.0729
20. Roweis, S., et al.: A Sticker-Based Model for DNA Computation. *J. Comp. Bio.* 5, 615–629 (1996)
21. Swartzlander, E.E., Alexopoulos, A.G.: The Sign/Logarithm Number System. *IEEE Trans. Comput.* C-24, 1238–1242 (1975)
22. Xu, J., Dong, Y., Wei, X.: Sticker DNA Computer Model-Part I: Theory. *Chin. Sci. Bull.* 49, 772–780 (2004)
23. Yang, X.Q., Liu, Z.: DNA Algorithm of Parallel Multiplication Based on Sticker Model. *Comp. Engr. App.* 43(16), 87–89 (2007)
24. Yurke, B., et al.: DNA Implementation of Addition in which the Input Strands are Separate from the Operator Strands. *Biosystem* 52, 165–174 (1999)

# Graph-Theoretic Formalization of Hybridization in DNA Sticker Complexes

Robert Brijder, Joris J.M. Gillis\*, and Jan Van den Bussche

Hasselt University and transnational University of Limburg

**Abstract.** Sticker complexes are a formal graph-based data model for a restricted class of DNA complexes, motivated by potential applications to databases. This data model allows for a purely declarative definition of hybridization. We introduce the notion of terminating hybridization, and characterize this notion in purely graph-theoretic terms. Terminating hybridization can still produce results of exponential size. We indicate a class of complexes where hybridization is guaranteed to be polynomially bounded.

## 1 Introduction

Since Adleman's experiment [2], DNA Computing has greatly evolved, and many different modes of computation have been invented and investigated [3, 21, 6, 25, 34, 26, 29, 14, 5, 33, 30, 32, 22]. A major goal throughout this evolution has been to achieve autonomy of computation, and indeed this is a highly desirable feature of computation in general.

At the same time, DNA Computing has also high potential for database applications [4, 10, 35, 24]. Indeed, the nanoscale and relative indestructibility of single DNA strands are very promising properties for database storage. Moreover, the highly parallel mode of operation that can be achieved in DNA Computing is a nice match with the bulk-processing nature of database computations.

Autonomy of computation is perhaps less crucial for databases, where indeed traditionally a strict line is drawn between the data, and the query or update operations performed on the data [15]. Also, in database theory [1], one expects formal data models defined on the logical level, and declarative definitions of the basic data manipulation operations.

In the present paper, in the context of a formal data model of DNA complexes, we focus on hybridization, one of the cornerstone operations in DNA computing. The data model is that of *sticker complexes*, a graph-theoretically defined formalization of DNA complexes of a limited format. Sticker complexes have been shown in an earlier paper [16] to be adequate for database computations in DNA. Indeed, while it is relatively straightforward to represent relational databases in DNA, a good data model for database computation must also be able to represent all intermediate data structures needed to support database operations.

---

\* Ph.D. Fellow of the Research Foundation Flanders (FWO).

Specifically, it has been shown that sticker complexes are adequate to support a complete simulation of the operations of the relational algebra, which provides a set of core operations in relational databases [15]. The intermediate data structures involved in the simulation of the relational algebra are quite complex as they need to support the creation of circular strands.

The problem addressed in the present paper is to understand the well-definedness and *termination* of the hybridization operation on sticker complexes. Here we are considering hybridization as a database operation, like the Cartesian product (related to the relational join). When we want to construct the Cartesian product  $U \times V$  of two sets  $U$  and  $V$ , with  $U$  of size  $m$  and  $V$  of size  $n$ , we need in principle  $n$  copies of every element of  $U$ , and  $m$  copies of every element of  $V$ , so that we have enough “material” to construct the  $m \times n$ -element set  $\{(u, v) \mid u \in U \ \& \ v \in V\}$ . When more copies are provided of some elements of  $U$  or  $V$ , some duplicate pairs can be constructed, but no really new information is generated. When hybridization has this behavior, we say it *terminates*.

The main result of this paper is to provide a purely graph-theoretic characterization of termination of hybridization, which will also imply that termination is decidable for sticker complexes. This result emphasizes the restricted nature of the sticker complex data model, since it is well known that termination is undecidable for Turing-universal computation models [18]. The investigation of computation models that are not computationally complete, and the corresponding search for the right balance between sufficient expressive power and low complexity, is one of the hallmarks of database theory [1].

We also investigate complexity issues related to DNA hybridization. Even when hybridization in a given DNA complex terminates, depending on the structure of the complex, an exponential amount of material may be required to produce the complete result. This problem was already present in Adleman’s solution to the Hamiltonian Path problem [17], and we show it can still occur within the limited context of sticker complexes. Since such exponential behavior is undesirable, and also not needed to support typical database operations, we would like to avoid it.

We will show that the result of hybridization splits up, graph-theoretically, in a number of connected components, and each component is polynomial in size. Hence, the exponentiality is confined to the possible number of distinct components. Furthermore, we identify a broad family of classes of DNA complexes, called *c-bounded complexes*, within which hybridization is guaranteed to require only a polynomial amount of resources.

## 2 Related Work

In one of the first papers on DNA computing, Reif already defined a formal data structure of DNA complexes [23]. Our data structures are simpler in an effort to avoid unrealistic or otherwise complicated and unmanageable secondary structures. (Reif avoids these by invoking an oracle for feasibility.) Our simplification is that single strands are either all-positive or all-negative, and moreover, negative strands have length at most two. The short negative strands can be thought

of as stickers; thus the name “sticker complexes”. Our previous work showed that the restrictions of sticker complexes do not preclude interesting database computations. An important feature of our model, which is lacking in Reif’s, is the formal distinction between the structural content of a complex, and the complex as used in reactions, with multiples of each connected component present in surplus quantities.

The use of short stickers in DNA computing originates with Roweis et al. [26], where stickers were used to turn bits on or off. We use stickers to bind strands together so that possibly complex secondary structures are formed.

The present work also fits in a recent trend of integrating formal methods (such as process calculi in computational systems biology [7]) with DNA computing [8, 20]. Yet the formalisms we use are different from process calculi and comprise mainly set theory, graph theory, and logic-based query languages. The computational power of hybridization in various models of formal languages has been intensively studied, e.g., [21, 34].

Last but not least, our formal model of hybridization is strikingly comparable in spirit to the model of Jonoska, McColm and Staninska [19]. That paper introduces the notion of a “pot type”; for the purpose of hybridization, the model of pot types and our model of sticker complexes are roughly equivalent. It was shown that *weak satisfiability* is decidable in polynomial time. Using our own terminology as introduced in Section 4, a complex is weakly satisfiable if it admits at least one “finished” component. In contrast, we show that *termination* is decidable in polynomial time. Termination is a stronger property than weak satisfiability; a terminating complex is weakly satisfiable, but a complex may be weakly satisfiable without terminating. (Jonoska et al. also consider stronger notions of satisfiability, but these do not relate to termination.)

### 3 The Sticker-Complex Data Model

From the outset we assume a finite alphabet  $\Sigma$ . As customary in formal models of DNA computing [21], each letter represents a *domain*, i.e., a string over the DNA alphabet  $\{A, C, G, T\}$ . The set of resulting domains must form a set of DNA codewords [11, 27, 31]. This should always be kept in mind.

The alphabet  $\Sigma$  is matched with its negative version  $\bar{\Sigma} = \{\bar{a} \mid a \in \Sigma\}$ , disjoint from  $\Sigma$ . Thus there is a bijection between  $\Sigma$  and  $\bar{\Sigma}$ , which is called *complementarity* and is denoted by overlining; we also set  $\bar{\bar{a}} = a$  so complementarity is symmetric. Obviously,  $\bar{a}$  stands for the Watson-Crick complement of the DNA sequence represented by  $a$ . The elements of  $\Sigma$  are called *positive symbols* and the elements of  $\bar{\Sigma}$  are called *negative symbols*.

We recall some fundamental definitions from our previous paper [16], suitably simplified according to the focus of the present paper. The simplifications are only for the purpose of presentation, and our results can be adapted to the original data model, which provides facilities for immobilizing and blocking specific pieces of a complex.

The overall structure of a DNA complex is abstracted in the notion of *pre-complex*. Formally, a pre-complex is a 4-tuple  $(V, L, \lambda, \mu)$  where

1.  $V$  is a finite set of nodes;
2.  $L \subseteq V \times V$  is a finite set of directed edges without self-loops (i.e.,  $(v, v)$  is not in  $L$  for all  $v \in V$ );
3.  $\lambda: V \rightarrow \Sigma \cup \bar{\Sigma}$  is a total function labeling the nodes;
4.  $\mu \subseteq \{\{v, w\} \mid v, w \in V \text{ and } v \neq w\}$  is a partial matching on the nodes, i.e., each node occurs in at most one pair in  $\mu$ . Note that the pairs in  $\mu$  are unordered.

Let  $C$  be a pre-complex as above. A *strand* of  $C$  is simply a connected component of the directed graph  $(V, L)$ , so ignoring  $\mu$ . The *length* of a strand is its number of nodes. A *sticker complex* now is a pre-complex satisfying the following restrictions:

1. Each node has at most one incoming and at most one outgoing edge. Thus, each strand has the form of a chain or a cycle.
2. Strands are homogeneously labeled, in the sense that either all nodes are labeled with positive symbols, or all with negative symbols. Naturally, a strand with positive (negative) symbols is called a positive (negative) strand.
3. Every negative strand has length one or two; if it has length two, then it must have a single edge (i.e., it cannot be a 2-cycle). Negative strands are also referred to as “stickers”.
4. Matchings by  $\mu$  only occur between complementarily labeled nodes: formally, if  $\{x, y\} \in \mu$  then  $\lambda(y) = \overline{\lambda(x)}$ .

In this way, the edges of a sticker complex indicate the sequence order within strands, and the matching  $\mu$  makes explicit where stickers have annealed to positive strands.

We will also refer to sticker complexes simply as “complexes”.

*Example 1.* A simple example of a complex is depicted in Fig. 1. The alphabet used is  $\{a, b, c\}$  with  $\bar{a}$ ,  $\bar{b}$  and  $\bar{c}$  indicated in the figure as  $A$ ,  $B$  and  $C$ , respectively. We will use this convention of showing complementary symbols by capitalizing the symbols, throughout the figures in this paper. The complex consists of ten nodes  $x_1, \dots, x_{10}$ , labeled as follows:

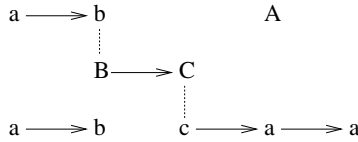
$$\begin{array}{l} \text{node } x : x_1 \ x_2 \ x_3 \ x_4 \ x_5 \ x_6 \ x_7 \ x_8 \ x_9 \ x_{10} \\ \text{label } \lambda(x) : a \ b \ \bar{a} \ \bar{b} \ \bar{c} \ a \ b \ c \ a \ a \end{array}$$

The nodes are organized in five strands: the negative strand  $\bar{a}$  of length 1, two copies of the positive strand  $ab$  of length 2; the negative strand  $\bar{b}\bar{c}$  of length 2; and the positive strand  $caa$  of length 3. More formally, we have

$$L = \{(x_1, x_2), (x_4, x_5), (x_6, x_7), (x_8, x_9), (x_9, x_{10})\}.$$

The matching  $\mu$  contains the two unordered pairs  $\{x_2, x_4\}$  and  $\{x_5, x_8\}$ .

*Remark 1.* Because stickers are short, there is no need in our model to require that annealed stickers run in complementary ( $5'-3'$  vs  $3'-5'$ ) directions with



**Fig. 1.** Example of a sticker complex. Capitalized letters A, B, and C denote complemented symbols  $a$ ,  $b$ , and  $c$ . The dotted lines denote the matching  $\mu$ .



**Fig. 2.** On the left of (i) and (ii), two different complexes; on the right of (i) and (ii), a depiction of their respective plausible realizations in DNA (recall that each node in the complex represents a DNA sequence, depicted here as thick blue lines)

respect to the positive strands they are annealed to. Indeed, for a sticker of length one, the complementarity is already built into the label; stickers of length two can fold so as to run in complementary direction. Fig. 2 gives an illustration.  $\square$

Note that, in a complex, not all nodes that can be matched must be matched: for example, in Fig. 1, the sticker  $\bar{a}$  is not annealed, but could anneal to the four different nodes labeled  $a$ . Indeed, it is the hybridization operation, defined below, that will perform all possible matchings.

*Components and redundancy.* We say that two strands  $s$  and  $s'$  in a complex are *bonded* if there exists some node  $v$  in  $s$  and some node  $v'$  in  $s'$  with  $\{v, v'\} \in \mu$ . When two strands are connected, possibly indirectly, by this bonding relation, we say they belong to the same component. Thus, a *component* of a pre-complex is a substructure formed by a maximal set of strands connected by the bonding relation. Put another way, whereas a *strand* was defined as a connected component ignoring  $\mu$ , a component is a connected component *not* ignoring  $\mu$ .

*Example 2.* The complex from Example 1 has three components: one consisting of the single strand  $\bar{a}$ , one consisting of the single strand  $ab$ , and one formed by the three strands  $ab$ ,  $\bar{b}\bar{c}$  and  $caa$ .  $\square$

The intention of our model is that a complex defines the structural content of a test tube. The test tube, however, will in practice hold copies in surplus quantity of each component. Thus, each component of a complex stands for possibly multiple occurrences. We formalize this intention using the notions of subsumption, equivalence, and minimality.

A complex  $C$  is said to *subsume* a complex  $C'$  if for each component  $D'$  of  $C'$ , there exists an component  $D$  in  $C$  that is isomorphic to  $D'$ . Two complexes



$C$  and  $C'$  are said to be *equivalent* if they subsume each other. A component  $D$  of a complex  $C$  is called *redundant* if some other component of  $C$  is isomorphic to  $D$ . Note that removing a redundant component from  $C$  yields a complex that is still equivalent to  $C$ .

*Remark 2.* Isomorphism of sticker complexes can be decided in polynomial time by depth-first search. Indeed, if  $C$  and  $C'$  both consist of a single component,  $v$  is a node of  $C$ , and  $v'$  is a node of  $C'$ , then there is at most one isomorphism from  $C$  to  $C'$  mapping  $v$  to  $v'$ , and this isomorphism can be traced out by depth-first search, following the chain or cycle shape of strands, and the partial matching  $\mu$ . Depth-first search is in linear time, which yields an isomorphism check for single components in cubic time (try all combinations of  $v$  and  $v'$ ). This algorithm then easily extends to complexes  $C$  and  $C'$  with multiple components, by matching the components of  $C$  to the components of  $C'$ . This efficient isomorphism check is in contrast to the problem of general graph isomorphism, which is not known to be decidable in polynomial time. We thus see that sticker complexes form a restricted family of graphs.

## 4 Hybridization

We give a purely declarative definition of hybridization, in a few steps. We define the two auxiliary notions of “hybridization extension” and “redundant variation”. This will allow us to define the fundamental notion of “multiplying hybridization extension (MHE)”. The final results of hybridization are then defined as the “saturated” MHEs; those that consist only of “finished” components.

Let  $C = (V, L, \lambda, \mu)$  and  $C' = (V', L', \lambda', \mu')$  be two complexes. We call  $C'$  a *hybridization extension* of  $C$  if  $V' = V$ ,  $L' = L$ ,  $\lambda' = \lambda$ , and  $\mu'$  is an extension of  $\mu$ , i.e.,  $\mu' \supseteq \mu$ . A complex  $C'$  is said to have *maximal matching* if the only hybridization extension of  $C'$  is  $C'$  itself.

*Example 3.* The complex from Example 1 does not have maximal matching; we can properly extend it by adding the pair  $\{x_3, x_9\}$  to  $\mu$ . Alternatively, instead of  $x_9$ , we could have taken  $x_1$ , or  $x_6$ , or  $x_{10}$ . Thus the complex has, apart from itself (which is a trivial hybridization extension), four different (non-equivalent) hybridization extensions. These four extensions all have maximal matching, since  $x_3$  is the only negatively labeled node that is not yet matched.  $\square$

Let  $C$  and  $C'$  again be complexes. We call  $C'$  a *redundant variation* of  $C$ , simply if  $C$  subsumes  $C'$ . Note that  $C'$  may contain redundant components. Hence, the recipe to produce a redundant variation is simply to take, for every component of  $C$ , zero, one, or more copies.

Hybridization is now defined in terms of *multiplying hybridization extensions (MHEs)*, which, by applying redundant variations, account for the presence of surplus copies of components participating in the hybridization. Let  $C$  and  $C'$  again be two complexes. We call  $C'$  an MHE of  $C$  if  $C'$  is a hybridization extension of some redundant variation  $C''$  of  $C$ .

The notion of MHEs is invariant under equivalence, both on the input side as on the output side:



**Fig. 3.** Finished MHE components for the complex shown in Fig. 1

**Proposition 1.** *Let  $C_1$  and  $C_2$  be two equivalent complexes.*

1. *A complex  $C'$  is an MHE of  $C_1$  if and only if  $C'$  is an MHE of  $C_2$ .*
2.  *$C_1$  is an MHE of a complex  $C$  if and only if  $C_2$  is an MHE of  $C$ .*

We are not quite finished with the notion of MHE, however. Indeed, an MHE may have “unfinished” components. Formally, we call a component  $D$  of an MHE *unfinished* if there exists another MHE in which  $D$  occurs bonded within a larger component; otherwise it is called *finished*. An MHE without any unfinished components is called *saturated*.

*Example 4.* None of the four hybridization extensions of the complex discussed in Example 3 is saturated. Indeed, as long as a component has an unmatched  $a$ , that component is unfinished because we can add a copy of the sticker  $\bar{a}$ . Specifically, we can finish the large component (consisting of the strands  $ab$ ,  $\bar{b}\bar{c}$ , and  $caa$ ) by matching each unmatched  $a$  to a fresh copy of  $\bar{a}$ , yielding the finished MHE component shown in Fig. 3 (left). Likewise we can finish the component consisting of the single strand  $ab$  by matching the  $a$  to a copy of  $\bar{a}$ , as shown in Fig. 3 (right). Finishing the component consisting of the single sticker  $\bar{a}$  can be done in two ways: by bringing in a copy of the large component, we get the same result as finishing that large component, and by bringing in a copy of the strand  $ab$ , we get the same result as finishing that strand. We conclude that there are precisely two distinct finished MHE components.

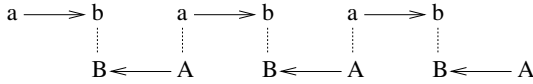
*Example 5.* A complex may have a large number of different finished MHE components: exponentially many in the size of the complex. For example, consider the complex  $C_n$  consisting of the following strands:

- a positive strand  $a \dots a$  of length  $n$  consisting of  $n$  nodes all labeled  $a$ ;
- a sticker  $\bar{a}\bar{b}$ ;
- a sticker  $\bar{a}\bar{c}$ .

Up to equivalence, there are precisely  $2^n$  finished MHE components for  $C_n$ . Each possibility is obtained by annealing, to each node of the positive strand, a copy of either the first or the second sticker.  $\square$

We finally define:

**Definition 1.** *Let  $C$  be a complex. The hybridization of  $C$  equals the disjoint union of all finished MHE components for  $C$ .*



**Fig. 4.** Illustration for Example 6

*Termination.* A fundamental issue regarding the above definition is that the result of hybridization as defined may be infinite, as shown next.

*Example 6.* Consider the simple complex consisting of two strands  $ab$  and  $\bar{b}\bar{a}$  and no matchings. For any number  $n$ , using  $n$  copies of  $ab$  and  $n$  copies of  $\bar{b}\bar{a}$ , we can produce the MHE component shown in Fig. 4 for  $n = 3$ . This component could also be finished, by matching the remaining  $a$  shown on the left with the remaining  $\bar{a}$  on the right, effectively creating a ring structure. (As always, in the figure,  $\bar{a}$  and  $\bar{b}$  are shown as  $A$  and  $B$ .) Different numbers  $n$  yield nonequivalent (non-isomorphic) MHE components, thus the number of potential MHE components is infinite.  $\square$

Nature will compute the result of hybridization by composing MHE's using the available material in the test tube. When, for a given complex  $C$ , there are actually infinitely many nonequivalent MHE's, we say that *hybridization does not terminate for  $C$* , or shorter, that  *$C$  is nonterminating*; otherwise, we say that *hybridization terminates*, or shorter, that  *$C$  is terminating*.

*Example 7.* So, the complex discussed in the previous example is nonterminating. In contrast, the example complex of Fig. 1 is terminating, as we have seen in Example 4. Also the complexes  $C_n$  discussed in Example 5 are terminating.  $\square$

In practice, when we have termination of hybridization, a test tube prepared with sufficient quantities of each component of the complex holds, in principle, sufficient material to produce all molecular species that can be the result of hybridization. If sufficient quantities are present, adding even more material will not yield new results. Of course, in practice, a test tube is always finite and the hybridization reaction will, under normal conditions, always “terminate” (reach equilibrium). But the point is that, when hybridization does not terminate for a complex, adding ever more material can, in principle, result in ever more new molecular species (MHE components) to be produced. In this sense, the potential result of the hybridization is indeed infinite.

## 5 Deciding Termination

When designing DNA complexes for DNA computing, it is of course highly desirable to recognize easily whether or not a given complex is terminating. Our main result is the following.

**Theorem 1.** *A complex is terminating if and only if its hybridization graph does not contain an alternating cycle.*

**Corollary 1.** *Termination of hybridization is decidable in polynomial time.*

We still need to define the relevant terms used in our theorem, i.e., “hybridization graph” and “alternating cycle”. The Corollary will follow since the hybridization graph has the same number of nodes as the given complex, and checking for the presence of an alternating cycle can be done in polynomial time.

The hybridization graph of a complex is an instance of a “partitioned graph”. A *partitioned graph* in general is a triple  $(V, \pi, E)$  where  $(V, E)$  is an undirected graph and  $\pi$  is a partition of the node set  $V$ . Recall that an undirected graph  $(V, E)$  consists of a set  $V$  of nodes and a set  $E \subseteq \{\{v, w\} \mid v, w \in V \text{ and } v \neq w\}$  of unordered pairs of nodes (undirected edges). Recall that a partition of a set  $V$  is a set of nonempty, pairwise disjoint subsets of  $V$ , called *blocks*, such that their union equals  $V$ .

Now given a complex  $C$ , the *hybridization graph for  $C$*  is the partitioned graph  $H = (V, \pi, E)$  defined as follows:

- $V$  equals the set of nodes of  $C$ ;
- $\pi$  contains, for each component  $D$  of  $C$ , the set of nodes belonging to  $D$  as a block;
- Let  $F \subseteq V$  be the set of “free” nodes of  $C$ ; a node is called *free* if it is not matched to another node by  $\mu$ . Then  $E$  equals  $\{\{v, w\} \mid v, w \in F \text{ and } \lambda(w) = \overline{\lambda(v)}\}$ .

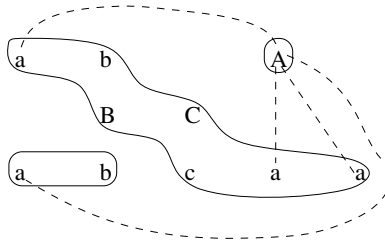
Thus, whereas the matching  $\mu$  in  $C$  represents the pairs of nodes that are *already* annealed, the set  $E$  contains the pairs of nodes that *may* still be annealed (typically, in an MHE of  $C$ ).

*Example 8.* The hybridization graph for the complex of Fig. 1 is shown in Fig. 5. The blocks are depicted as hyperedges (closed curves enclosing the nodes belonging to the same block). The undirected edges are shown as dashed lines.  $\square$

The notion of alternating cycle can be defined in general in any partitioned graph  $G = (V, \pi, E)$ . A *path* in  $G$  is a sequence of nodes  $v_1, \dots, v_n$  such that for each  $i$  with  $1 \leq i < n$ , we have either an

**edge move:**  $\{v_i, v_{i+1}\} \in E$ , or a

**block move:**  $v_i \neq v_{i+1}$  and they belong to a common block.



**Fig. 5.** Example of a hybridization graph

The path is said to be *alternating* if edge moves happen for each odd  $i$ , and block moves happen for each even  $i$  (always for  $1 \leq i < n$ ). When the path is alternating, it is said to be an *alternating cycle* when  $n$  is odd and at least 3, and  $v_n = v_1$ .

*Example 9.* Consider the hybridization graph for the complex of Fig. 1, as shown in Fig. 5. We refer to the node identifiers given in Example 1. Two examples of alternating paths are  $p_1 = x_3, x_9, x_1, x_3$  and  $p_2 = x_3, x_1, x_{10}, x_3, x_6, x_7$ . Note that  $p_1$  is not an alternating cycle; although it satisfies  $v_n = v_1$ , its length, 4, is not odd. Indeed, this hybridization graph does not admit an alternating cycle, since the only free node with a negative label,  $\bar{a}$ , is in a component by itself.

*Example 10.* Consider the complex discussed in Example 6. Its hybridization graph has four nodes partitioned in two blocks. One block, corresponding to the component  $ab$ , consists of two nodes  $x_1$  and  $x_2$  labeled  $a$  and  $b$ , respectively; the second block, corresponding to the component  $\bar{b}\bar{a}$ , consists of two nodes  $y_1$  and  $y_2$  labeled  $\bar{b}$  and  $\bar{a}$ , respectively. There are two undirected edges, namely,  $\{x_1, y_2\}$  and  $\{x_2, y_1\}$ . This hybridization graph admits an alternating cycle in the form of  $x_1, y_2, y_1, x_2, x_1$ .  $\square$

The above two examples are in line with Theorem 1. Indeed, the complex of Fig. 1 is terminating, and indeed its hybridization graph does not have an alternating cycle; the complex of Example 6 is nonterminating, and indeed its hybridization graph has an alternating cycle.

The only-if implication of Theorem 1 is relatively easy to prove. The proof of the if-implication (omitted) involves a constructive characterization of MHE components in the form of “hybridization templates”, which we present here.

We first need the following auxiliary notion. Let  $G = (V, E)$  and  $G' = (V', E')$  be two undirected graphs, and let  $f : V \rightarrow V'$  be a mapping. Then  $f$  is called a *semi-strong homomorphism from  $G$  to  $G'$*  if, for all  $u, v \in V$ , we have the following:

- if  $\{u, v\} \in E$  then  $\{f(u), f(v)\} \in E'$ ; and
- if  $\{f(u), f(v)\} \in E'$  then  $\{u, w\} \in E$  for some  $w \in V$ , or  $\{v, w\} \in E$  for some  $w \in V$ .

The first condition is the standard requirement for homomorphisms; the converse of that condition would state the standard requirement for what is known in universal algebra as a “strong” homomorphism. The second condition, however, states only a weak converse (hence the name “semi-strong”), in the sense that if there is an edge between  $f(u)$  and  $f(v)$ , then either  $u$  or  $v$  have to be involved in an edge, but not necessarily with each other.

Now let  $C = (V, L, \lambda, \mu)$  be a complex with hybridization graph  $H = (V, \pi, E)$ . A *hybridization template for  $C$*  is a pair  $T = (t, f)$  where  $t = (V^t, \pi^t, E^t)$  is a partitioned graph and  $f$  is a semi-strong homomorphism from  $(V^t, E^t)$  to  $(V, E)$ , such that:

1.  $t$  is connected, i.e., there is a path between any two distinct nodes (using the notion of path in partitioned graphs as defined earlier);

2.  $E^t$  is a partial matching, i.e., each node of  $V^t$  occurs in at most one edge in  $E^t$ ; and
3. for each block  $q$  of  $\pi^t$  there is a block  $q'$  of  $\pi$  such that the restriction  $f|_q$  of  $f$  to  $q$  is a bijection from  $q$  to  $q'$ , i.e.,  $f|_q$  is injective and the image of  $f|_q$  equals  $q'$ .

From a hybridization template  $T = (t, f)$  for  $C$ , and  $C$  itself, we can construct a sticker complex  $\text{comp}(T) = (V^T, L^T, \lambda^T, \mu^T)$  as follows:

- $V^T = V^t$ ;
- $L^T = \{(x, y) \mid x \text{ and } y \text{ belong to a common block and } (f(x), f(y)) \in L\}$ ;
- $\lambda^T(x) = \lambda(f(x))$ ;
- $\mu^T = E^t \cup \{(x, y) \mid x \text{ and } y \text{ belong to a common block and } \{f(x), f(y)\} \in \mu\}$ .

**Proposition 2.** *The MHE components are exactly the complexes of the form  $\text{comp}(T)$  with  $T$  a hybridization template.*

The proof of Theorem 1 also invokes the following lemma which may be interesting in its own right:

**Lemma 1.** *Let  $H$  be a partitioned graph with  $c$  distinct blocks. If  $H$  admits no alternating cycle, then the length of any alternating path in  $H$  is at most  $4c + 2$ .*

## 6 Complexity Issues

Assume hybridization terminates for a given sticker complex  $C$ . Then two follow-up questions come up related to the complexity of the result of hybridization. How many finished MHE components can there be? And, how large can a single finished MHE component become?

As we have already seen in Example 5, the *number* of finished MHE components may well grow exponentially in the size of the complex. Also the *size* of MHE components can grow exponentially (details omitted). Unlike Example 5, however, the latter can only happen when the alphabet is allowed to grow with the size of the complex. Usually, however, the alphabet is fixed by the application setting. Indeed we show: (proof omitted)

**Proposition 3.** *Over the class of terminating complexes over any fixed alphabet, the size of the largest MHE component for a complex  $C$  grows only polynomially in the size of  $C$ .*

Interestingly, the proof of this proposition relies on the following counterpart to Lemma 1. The two lemmas are complementary as Lemma 1 does not assume anything about the alphabet, whereas Lemma 2 does not assume anything about the complex.

**Lemma 2.** *Let  $H$  be the hybridization graph of a complex over positive alphabet  $\Sigma$ . Let  $s$  be the number of symbols in  $\Sigma$ . If  $H$  admits no alternating cycle, then the length of any alternating path in  $H$  is at most  $8s + 2$ .*

*Remark 3.* Since the number of possible graphs on a polynomial number of nodes is singly-exponential, as a corollary to Proposition 3, we obtain that over the class of terminating complexes over a fixed alphabet, the number of MHE components for a complex  $C$  is bounded from above by  $2^{n^{O(1)}}$ , where  $n$  is the size of  $C$ . Hence, Example 5 essentially illustrates the worst that can happen, i.e., double-exponential or worse is impossible.  $\square$

Our final result presents a restriction on classes of complexes, which we call “ $c$ -bounded choice” (for a natural number  $c$ ), so that hybridization is polynomial on the class of  $c$ -bounded complexes. It remains to be investigated further how practicable this restriction is, i.e., how many applications can be modeled using sticker complexes that are  $c$ -bounded for some  $c$ . A positive indication is that only 4-bounded complexes are needed to simulate the relational algebra; to verify this we have inspected the procedures given in an earlier paper [16].

To define the notion of  $c$ -boundedness, we first need the notion of a “choice node” of a complex. This is a free node having at least two neighbors in the hybridization graph. Since the edges of the hybridization graph are solely defined in terms of free nodes and their labels being complementary, we see the following, for any label  $a \in \Sigma \cup \bar{\Sigma}$ : a node  $v$  labeled  $a$  is a choice node if and only if it is free and there exist at least two free nodes labeled  $\bar{a}$ . Consequently, if there are at least two free nodes labeled  $\bar{a}$ , then *all* free nodes labeled  $a$  are choice nodes; in the other case, *no* node labeled  $a$  is a choice node.

Now for any natural number  $c$ , we say that a complex  $C$  has  *$c$ -bounded choice*, or shorter, *is  $c$ -bounded*, if for each component  $D$  of  $C$ , the number of choice nodes reachable by alternating paths from any node in  $D$ , is at most  $c$ . Here, naturally, we say that a node  $w$  is reachable by an alternating path from a node  $v$ , if there is an alternating path starting with  $v$  and ending with  $w$ . In particular, any node is reachable from itself by an alternating path, since the length-one path  $v$  is a trivial alternating path.

*Example 11.* Recall the complexes  $C_n$  discussed in Example 5. Recall that the number of finished MHE components for  $C_n$  is  $2^n$ . Since there are two free  $\bar{a}$ -nodes, the  $n$  nodes labeled  $a$  are all choice nodes. As these  $n$  nodes all belong to a common component, the smallest  $c$  such that  $C_n$  is  $c$ -bounded is  $n$ . Hence, there is no fixed  $c$  such that all  $C_n$ , for all  $n$ , are  $c$ -bounded.

Suppose now, we modify  $C_n$  to  $C'_n$  by removing the sticker  $\bar{a}\bar{c}$ . Then the  $a$ -nodes are no longer choice nodes. The only remaining choice node  $C'_n$  is the  $\bar{a}$ -labeled node. Hence, each  $C'_n$  is 1-bounded. Now note that each  $C'_n$  has only one finished component, obtained by annealing each  $a$ -node to the  $\bar{a}$ -node of a fresh copy of the sticker  $\bar{a}\bar{b}$ . In particular, hybridization is not exponential on the class of  $C'_n$  complexes for all  $n$ .  $\square$

The above example illustrates our result: (proof omitted)

**Theorem 2.** *Let  $c$  be a natural number. Over the class of terminating,  $c$ -bounded complexes over a fixed alphabet, the hybridization of any complex  $C$  has size polynomial in the size of  $C$ .*

*Remark 4.* Theorem 2 states that for  $c$ -bounded terminating complexes over a fixed alphabet, the result of hybridization has polynomial size. By Definition 1 and Proposition 3, this is the same as saying that the number of finished MHE components is polynomial. Note that it is *not* true that the number of *unfinished* MHE components is polynomial. For example, for each number  $n$ , consider a complex  $U_n$  with two components: one is the strand  $a \dots a$  ( $n$  times), and the other is the sticker  $\bar{a}$ . There are  $2^n - 1$  unfinished MHE components, by choosing a strict subset of the  $n$  positive nodes, and annealing to each of them a copy of the sticker. There is, however, a unique finished MHE component, obtaining by annealing a copy of the sticker to *all* positive nodes.

*Remark 5.* There is no converse to Theorem 2 in the sense that, if the result of hybridization has polynomial size over some class  $K$  of complexes over some fixed alphabet, then the complexes in  $K$  must be  $c$ -bounded for some fixed  $c$ . Take, for example, the class  $K$  consisting of all complexes  $L_n$ , for every number  $n$ , where  $L_n$  consists of four components: a strand  $d \dots d$  of length  $2^n$ ; a strand  $a \dots a$  of length  $n$ ; and two stickers  $\bar{a}\bar{b}$  and  $\bar{a}\bar{c}$ . The size of  $L_n$  is  $2^n + n + 4$ , and there are  $2^n$  finished MHE components for  $L_n$ , which is a number polynomial in the size of  $L_n$ . Yet, the class  $K$  is not  $c$ -bounded for any fixed  $c$ , since  $L_n$  contains  $n$  choice nodes.

## 7 Conclusion

A natural extension of our approach would be to account for probabilities or error rates on the results produced (finished or unfinished) during hybridization. Of course, error modeling in DNA computation, and secondary structure prediction, are well-known research problems, e.g., [13, 9].

In previous work [16] two of us have defined a database-oriented DNA programming language, called DNAQL, with the goal of understanding the database side of DNA computing. Various open problems remain in connection with this language, including guaranteeing well-definedness through a type system, and understanding the expressive power.

Obviously, we would also like to see the sticker complex data model justified physically (or understand what are the unrealistic aspects), either experimentally or by simulation.

**Acknowledgement.** We thank the program committee for referring us to the work of Jonoska et al. [19].

## References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley, Reading (1995)
2. Adleman, L.: Molecular computation of solutions to combinatorial problems. Science 226, 1021–1024 (1994)
3. Amos, M.: Theoretical and Experimental DNA Computation. Springer, Heidelberg (2005)



4. Arita, M., Hagiya, M., Suyama, A.: Joining and rotating data with molecules. In: Proceedings 1997 IEEE International Conference on Evolutionary Computation, pp. 243–248 (1997)
5. Benenson, Y., Gil, B., Ben-Dor, U., Adar, R., Shapiro, E.: An autonomous molecular computer for logical control of gene expression. *Nature* 429, 423–429 (2004)
6. Boneh, D., Dunworth, C., Lipton, R., Sgall, J.: On the computational power of DNA. *Discrete Applied Mathematics* 71, 79–94 (1996)
7. Cardelli, L.: Abstract machines of systems biology. In: Priami, C., Merelli, E., Gonzalez, P., Omicini, A. (eds.) *Transactions on Computational Systems Biology III*. LNCS (LNBI), vol. 3737, pp. 145–168. Springer, Heidelberg (2005)
8. Cardelli, L.: Strand algebras for DNA computing. In: Deaton and Suyama [12], pp. 12–24
9. Chen, H.L., Kao, M.Y.: Optimizing tile concentrations to minimize errors and time for DNA tile self-assembly systems. In: Sakakibara and Mi [28], pp. 13–24
10. Chen, J., Deaton, R., Wang, Y.Z.: A DNA-based memory with in vitro learning and associative recall. *Natural Computing* 4(2), 83–101 (2005)
11. Condon, A., Corn, R., Marathe, A.: On combinatorial DNA word design. *Journal of Computational Biology* 8(3), 201–220 (2001)
12. Deaton, R., Suyama, A. (eds.): *DNA 15*. LNCS, vol. 5877. Springer, Heidelberg (2009)
13. Dimitrov, R., Zuker, M.: Prediction of hybridization and melting for double-stranded nucleic acids. *Biophysical Journal* 87, 215–226 (2004)
14. Dirks, R., Pierce, N.: Triggered amplification by hybridization chain reaction. *Proceedings of the National Academy of Sciences* 101(43), 15275–15278 (2004)
15. Garcia-Molina, H., Ullman, J., Widom, J.: *Database Systems: The Complete Book*. Prentice-Hall, Englewood Cliffs (2009)
16. Gillis, J., Van den Bussche, J.: A formal model of databases in DNA. In: Horimoto, K., Nakatsui, M., Popov, N. (eds.) *Algebraic and Numeric Biology 2010*. LNCS, Springer, Heidelberg (to appear, 2011) for a preprint, <http://alpha.uhasselt.be/~vdbuss/dnaql.pdf>
17. Hartmanis, J.: On the weight of computations. *Bulletin of the EATCS* 55, 136–138 (1995)
18. Hopcroft, J., Ullman, J.: *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading (1979)
19. Jonoska, N., McColm, G., Staninska, A.: On stoichiometry for the assembly of flexible tile DNA complexes. *Natural Computing*, January 23 (2010) (published online)
20. Majumder, U., Reif, J.: Design of a biomolecular device that executes process algebra. In: Deaton and Suyama [12], pp. 97–105
21. Paun, G., Rozenberg, G., Salomaa, A.: *DNA Computing*. Springer, Heidelberg (1998)
22. Qian, L., Soloveichik, D., Winfree, E.: Efficient Turing-universal computation with DNA polymers. In: Sakakibara and Mi [28], pp. 123–140.
23. Reif, J.: Parallel biomolecular computation: models and simulations. *Algorithmica* 25(2-3), 142–175 (1999)
24. Reif, J.H., LaBean, T.H., Pirrung, M., Rana, V.S., Guo, B., Kingsford, C., Wickham, G.S.: Experimental construction of very large scale DNA databases with associative search capability. In: Jonoska, N., Seeman, N.C. (eds.) *DNA 2001*. LNCS, vol. 2340, pp. 231–247. Springer, Heidelberg (2002)

25. Rothemund, P.: A DNA and restriction enzyme implementation of Turing machines. In: Lipton, R., Baum, E. (eds.) DNA Based Computers: DIMACS Workshop, held April 4, pp. 75–120. American Mathematical Society, Providence (1996)
26. Roweis, S., Winfree, E., Burgoyne, R., Chelyapov, N., Goodman, M., Rothemund, P., Adleman, L.: A sticker-based model for DNA computation. *Journal of Computational Biology* 5(4), 615–629 (1998)
27. Sager, J., Stefanovic, D.: Designing nucleotide sequences for computation: A survey of constraints. In: Carbone, A., Pierce, N.A. (eds.) DNA 2005. LNCS, vol. 3892, pp. 275–289. Springer, Heidelberg (2006)
28. Sakakibara, Y., Mi, Y. (eds.): DNA 16 2010. LNCS, vol. 6518. Springer, Heidelberg (2011)
29. Sakamoto, K., et al.: State transitions by molecules. *Biosystems* 52, 81–91 (1999)
30. Seelig, G., Soloveichik, D., Zhang, D., Winfree, E.: Enzyme-free nucleic acid logic circuits. *Science* 315(5805), 1585–1588 (2006)
31. Shortreed, M., et al.: A thermodynamic approach to designing structure-free combinatorial DNA word sets. *Nucleic Acids Research* 33(15), 4965–4977 (2005)
32. Soloveichik, D., Seelig, G., Winfree, E.: DNA as a universal substrate for chemical kinetics. In: PNAS 2010, March 4 (2010) (published online)
33. Soloveichik, D., Winfree, E.: The computational power of Benenson automata. *Theor. Comput. Sci.* 244(2–3), 279–297 (2005)
34. Winfree, E., Yang, X., Seeman, N.: Universal computation via self-assembly of DNA: Some theory and experiments. In: Landweber, L., Baum, E. (eds.) DNA Based Computers II: DIMACS Workshop, held June 10–12, pp. 191–213. American Mathematical Society, Providence (1998)
35. Yamamoto, M., Kita, Y., Kashiwamura, S., Kameda, A., Ohuchi, A.: Development of DNA relational database and data manipulation experiments. In: Mao, C., Yokomori, T. (eds.) DNA12. LNCS, vol. 4287, pp. 418–427. Springer, Heidelberg (2006)

# Localized Hybridization Circuits

Harish Chandran<sup>1</sup>, Nikhil Gopalkrishnan<sup>1</sup>, Andrew Phillips<sup>2</sup>, and John Reif<sup>1</sup>

<sup>1</sup> Department of Computer Science, Duke University

{harish,nikhil,reif}@cs.duke.edu

<sup>2</sup> Microsoft Research

Andrew.Phillips@microsoft.com

**Abstract.** Molecular computing executed via local interactions of spatially contiguous sets of molecules has potential advantages of (i) speed due to increased local concentration of reacting species, (ii) generally sharper switching behavior and higher precision due to single molecule interactions, (iii) parallelism since each circuit operates independently of the others and (iv) modularity and scalability due to the ability to reuse DNA sequences in spatially separated regions. We propose detailed designs for local molecular computations that involve spatially contiguous molecules arranged on addressable substrates. The circuits act via enzyme-free DNA hybridization reaction cascades. Our designs include composable OR, AND and propagation Boolean gates, and techniques to achieve higher degree fan-in and fan-out. A biophysical model of localized hybridization reactions is used to estimate the effect of locality on reaction rates. We also use the Visual DSD simulation software in conjunction with localized reaction rates to simulate a localized circuit for computing the square root of a four bit number.

## 1 Introduction

*Molecular computation* (MC) is computation executed at the molecular scale. Since the seminal work of Adleman[1], many DNA based computation schemes have been proposed.

We will be discussing MCs that use strands of DNA to store state and execute various reactions that involve DNA hybridization and DNA strand displacement processes. DNA hybridization is the non-covalent binding of two complementary DNA sequences to form a single duplex structure. DNA strand displacement is the displacement of a single strand of DNA from a double helix by an incoming strand with a longer complementary region to the template strand. The incoming strand has a toehold, an empty single stranded region on the template strand complementary to a subsequence of the incoming strand, to which it binds initially. It eventually displaces the outgoing strand via a kinetic process modeled as a one-dimensional random walk. Strand displacement is a key process in DNA nanoscience and many DNA nanosystems relying on DNA strand displacement have been demonstrated, ranging from DNA walkers[2] to catalytic circuits[3][4] to molecular detectors[5].

## 1.1 Local versus Global MC

We distinguish two types of MC:

- A MC is local if its state is encoded by a spatially contiguous set of molecules i.e., the state of each computing element is explicitly determined by the configuration of these molecules, and transitions of state are executed via interactions between local computing elements. An example of a purely local MC is whiplash PCR of Sakamoto et al.[6] in which the computation state is present within a single molecule, and state transitions are executed via polymerization reactions on that molecule. This paper gives a detailed description of a novel local MC using only DNA hybridization and no enzymatic reactions.

- In contrast, a MC is global if the state of the device is spatially distributed i.e., the state is determined by averaging the concentration and configuration of spatially distributed molecules that define state, and transitions of state are executed via multiple distributed interactions. An example of global MC is DNA reaction networks such as seesaw gates of Qian and Winfree[7] where all elementary logical operations (e.g., logical AND and OR) are performed via a host of diffusion based strand displacement interactions.

Note that other MC may combine local and global; for example the tile assembly of Rothemund and Winfree[8] performs computation by incorporating molecules into a growing assembly.

## 1.2 Motivation for Local MC

We discuss some potential advantages of local MC over global MC by also considering conventional computing devices. Decades of ingenious improvements (e.g., digital behavior, locality of memory, and local parallelism) to the design of conventional computing devices have allowed their performance to be vastly improved, providing optimizations beyond the improvements due purely to fabrication technology. We aim in this paper to exploit key aspects of these designs in our designs of local MCs.

**Execution Speedup via Locality of Memory:** In conventional computing devices such as digital processors data is stored locally wherever possible and only when the local memory limitations are exhausted is memory repositioned in more distributed locations. Hence conventional computing devices exploit locality as a critical method to improve execution time for both memory access and processing. It is reasonable that locality also be critical to the design of MC devices.

In local MC, because the dominant interactions of a molecule involve a fixed set of neighbors, we can preferentially speed up designed interactions over spurious ones. In contrast, in a global MC, such as a DNA hybridization circuit, the speed of execution is severely limited by diffusion processes, which are stochastic processes whose rates are governed by the mobility and concentration of the molecules. In particular, the rate of hybridization of two complementary freely floating DNA strands (of moderate length) in a dilute buffer solution is limited

by the frequency of collisions between these DNA strands, since these molecules must first find each other by the much slower diffusion reaction prior to the subsequent hybridization reaction that occurs after they contact each other[9], [10]. The rate of diffusion can be adjusted, but is ultimately limited by pragmatic considerations. Note that in a global MC, such as a DNA hybridization circuit, the mobility of molecules can be increased by increasing the temperature, but this is limited to be at most the melting temperature of the hybridization. Alternately, one can attempt in a global MC to increase the rate of collisions by increasing the concentration of the molecules, but this also increases the rate of spurious reactions (e.g., unintended hybridization and/or dehybridizations, etc), by the same factor. Hence, ultimately the computation rate of such global MCs is limited by diffusion rates, not the local reaction rates. This implies that global MCs, such as DNA hybridization networks which rely on diffusion at each step, may be inherently slower than local MCs where the local concentration of the reacting species results in collision rates that are several orders of magnitude larger.

**Digital Behavior in Conventional Computing Devices:** In conventional digital computing machines the digital behavior of switches is a fundamental building block and it is reasonable that they would also be critical to the design of MC devices. Bi-stable devices that are used to simulate switches exhibit analog behavior as they execute switching transitions. A sharper transition leads to faster switching times. Greater activation energies for the switching transition lead to increased robustness of the device. These two factors together decide the digital behavior of the device. This sharp digital behavior seems likely to be useful in incorporating variants of techniques (such as fault tolerance) perfected for standard silicon based digital devices into the design of MC devices. There are considerable differences in the digital behavior of local and global MC:

- Circuits implemented by local MC using spatially contiguous molecules exhibit sharp switching behavior within the computing unit. In particular, the state of each computing element is explicitly determined by the configuration of these molecules, so is unambiguous and digital in nature.
- In contrast, circuits implemented by global MC such as DNA hybridization reaction networks exhibit analog behavior that can be translated into digital behavior only by applying thresholds. In particular, the state of a global MC is determined by averaging the concentration and configuration of spatially distributed molecules that define state, and so is an analog value.

**Local Parallelism in Conventional Computing Devices:** Modern-day conventional computing devices exploit parallel processing at many levels, both at the local processor chip as well as in multi-core architectures, to allow computations to progress along different computational paths in parallel. MC can exploit local parallelism to allow multiple distinct local MCs to progress along different computational paths in parallel. For example, a plethora of programmable nanomanufacturing systems have been demonstrated[11][12]. These nanomanufacturing systems require the state information to be stored locally so that different products can be assembled in parallel in different molecules. Even if only a single

product is targeted, different instances of the same product might be at different stages of manufacture at any given time. This requires the state information to be stored locally and is ideally suited for local molecular computing. In contrast, it is not evident how to implement this local control in a global MC.

**Modularity:** In conventional computing, devices generally contain multiple identically designed computational units that act as modules in the overall architecture and considerably simplify the overall design. In MC, we can expect that modularity will also be a critical aspect of the design and in large part relates to DNA sequence reusability:

- Local MC involves interactions only with a fixed set of neighbors of each molecule and this enables distinct namespaces across molecules and opens up possibilities of sequence reuse in the system. In particular, since our local DNA hybridization circuits are spatially separated and cannot directly interact, we can reuse the same DNA sequences to build the same functionality (e.g., DNA hybridization circuit for a given logical operation) on a different part of the system using very few distinct DNA sequences in the overall design.
- Global MC on the other hand involves interactions among DNA strands that can be present anywhere in the reacting vessel, which implies a single global namespace for the sequences, and hence considerably limits DNA sequence reusability.

### 1.3 Our Contribution and Paper Organization

We develop detailed designs to implement DNA circuits on fully addressable DNA nanostructures such as a fully addressable lattice developed by Yan et al.[13] or DNA origami developed by Rothmund[14]. In doing so we develop a local molecular computing methodology to compute arbitrary Boolean functions. Our circuits are designed carefully to place downstream gates close enough to upstream gates to implement rapid signal transduction but far enough to minimize leaks. We argue that our circuits will: (i) be faster than chemical reaction networks due to increased local concentration of reacting species, (ii) exhibit generally sharper switching behavior and higher precision due to single molecule interactions, (iii) be highly parallel since each circuit operates independently of the others which finds use in nanomanufacture (iv) be modular and scalable due to the ability to reuse DNA sequences in spatially separated regions. We estimate the effect of locality on reaction rates via a biophysical model of localized hybridization reactions. We then use the Visual DSD simulation software in conjunction with these localized reaction rates to simulate a localized circuit for computing the square root of a four bit number.

### 1.4 Prior Work

In theory it is possible to perform arbitrary computations using DNA strands, ignoring issues of errors, scale and time. Current experimental demonstrations are far from this ideal. Synthetically designed DNA nanosystems currently cannot be leveraged to perform computationally complex biochemical tasks. The

need is for DNA circuits that are autonomous, error-resilient, scalable, fast and energy efficient. We review progress toward this goal over the past decade.

**Catalytic DNA Circuits:** Zhang et al.[3] argued that autonomous, scalable circuits require signal amplification and proposed catalysis as a means of achieving amplification. Their entropy-driven catalytic gate achieved an auto-catalytic exponential amplification of a DNA signal, as opposed to linear amplification[5]. The free energy for this reaction is a result of entropic gain due to dissociation of weakly-bound DNA strands. At the same time, Yin et al.[4] also illustrated the power of catalysis by demonstrating catalyzed formation of branched structures, auto-catalytic exponential amplifiers, dendritic growth and bipedal walkers. All these reactions were executed using an elementary hairpin motif with distinct functional sequence domains, allowing the reaction network to be abstracted by a simple visual symbolic language. These papers establish catalysis as a robust paradigm for constructing amplifying DNA gates.

**Composable DNA Gates:** The next challenge is to integrate catalytic DNA gates into large circuits. Qian and Winfree[15] proposed the simple and elegant *seesaw* gate, a modification of the catalytic gate of Zhang et al.[3] to make it composable. All toehold domains in the seesaw gate are identical, allowing for modular design of the circuit and parallel synthesis of many gates. Using the seesaw gates, Qian and Winfree[7] were able to demonstrate a non-trivial computation using hundreds of gates. The key obstacle toward further scaling up their circuit is the speed of operation. Since the seesaw gates are freely floating in solution, the reaction rates are primarily limited by the time it takes for DNA strands to encounter each other via diffusion. The authors take note of this issue and suggest moving their circuits to origami to speed-up hybridization kinetics, avoid cross talk and re-use sequences in spatially separate locations. This work expands upon those ideas.

**Two Domain Fork and Join Gates:** Cardelli[16] developed designs for transduction, fork and join gates using DNA strands limited to two domains. He analyzed the power of systems composed of these 3 basic gates, proved that they are equivalent to Petri nets and simulated various systems specified in this two domain language. Our independently derived designs for AND, OR and PROPAGATION gates are similar to the fork and join gates described in this work.

**Addressable DNA Substrates:** We develop designs that implement DNA hybridization circuits on fully addressable substrates. Much experimental progress has been achieved in the area of addressable DNA substrates. Park et al.[17] demonstrated a fully addressable lattice built out of the cross tile developed by Yan et al.[13]. This was later extended to an  $8 \times 8$  fully addressable lattice by Pistol and Dwyer[18] using hierarchical assembly techniques. Rothemund[14] developed the technique of DNA origami in which a long scaffold DNA strand obtained from a viral genome is folded into the desired shape by the use of hundreds of short synthetic DNA strands called staples. Each staple strand binds to the scaffold strand at precisely two locations thereby localizing two distinct points on the scaffold strand. Hundred such stapling events fold the scaffold

into the desired shape. Origami has been widely used as a substrate to arrange various molecules[12][19] and has been extended to form three dimensional shapes[20][21].

## 2 Design of DNA Hybridization Circuits on Addressable DNA Substrates

We begin by designing two composable DNA hybridization driven gates that perform AND and OR Boolean logic. An additional propagation gate serves as a wire and propagates signal. The gates are implemented as a cascade of toehold mediated strand displacement reactions. Each of these reactions is initiated via a *universal toehold binding domain* (labeled  $\tilde{T}$ ) whose sequence is the same throughout the circuit. The specificity of strand displacement is conferred by a set of *specificity domains* (labeled  $\tilde{S}_i$ ) that are unique to each gate. We assume that there exists an irreversible downstream drain (not shown in the figures) for each gate.

The gates can be precisely positioned on the fully addressable DNA substrate by designing them as extensions of conventional substrate strands. The actual positioning of the gates depends on the digital circuit being implemented. In particular, gates that are connected to each other are placed close to each other. In this sense, the localized DNA circuit mimics the topology of the actual digital circuit diagram. Each localized circuit structure contains one copy of each gate in the circuit and operates by signaling across gates via single molecules, eliminating the need for signal restoration.

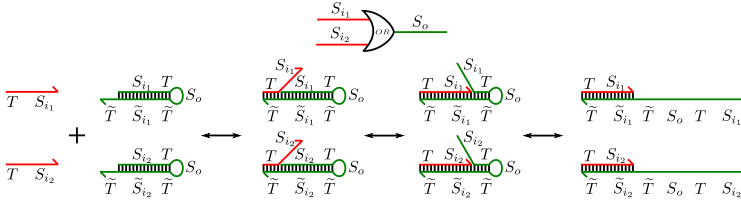
Any boolean function can be implemented using purely AND and OR gates by the use of dual rail logic, which uses two bits each to encode 0 and 1. The propagation gate is used to control the relative positions of the AND and OR gates on the substrate so that the circuit elements can function correctly without steric interference.

### 2.1 Design of Logical Gates

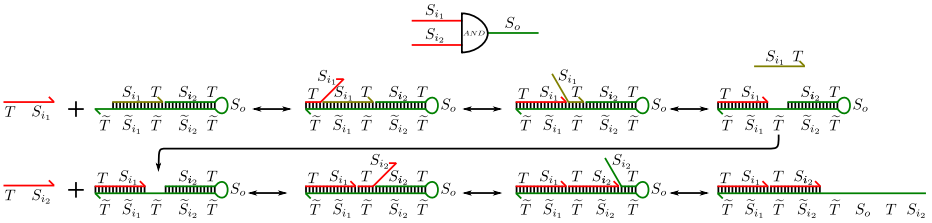
Figure 1 illustrates how two hairpin motifs are used to achieve OR logic. For input domains  $S_{i_1}$  and  $S_{i_2}$  we design motifs with the sequences  $S_{i_1}TS_o\tilde{T}\tilde{S}_{i_1}\tilde{T}$  and  $S_{i_2}TS_o\tilde{T}\tilde{S}_{i_2}\tilde{T}$  respectively where  $\tilde{S}_{i_1}$  and  $\tilde{S}_{i_2}$  are the input recognition domains and  $S_o$  is the output domain. In the presence of either the sequence  $TS_{i_1}$  or  $TS_{i_2}$ , one of the hairpins opens up to reveal the output  $S_o$  as illustrated in Figure 1. Note that the reaction that opens the hairpin is irreversible in the presence of a downstream gate that consumes  $S_o$ .

Figure 2 illustrates a two input AND gate complex consisting of a hairpin motif and a protector strand. For input domains  $S_{i_1}$  and  $S_{i_2}$ , the hairpin motif has sequence  $S_{i_2}TS_o\tilde{T}\tilde{S}_{i_2}\tilde{T}\tilde{S}_{i_1}\tilde{T}$  and is hybridized to the protector that has sequence  $S_{i_1}T$ . The sequences  $\tilde{S}_{i_1}$  and  $\tilde{S}_{i_2}$  are the input recognition domains and  $S_o$  is the output domain. In the presence of the sequence  $TS_{i_1}$  the protector is displaced out of the complex exposing a universal toehold domain  $\tilde{T}$ . If the

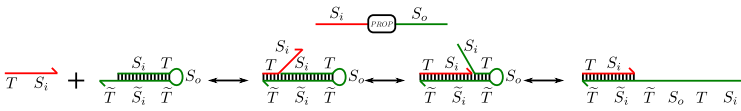




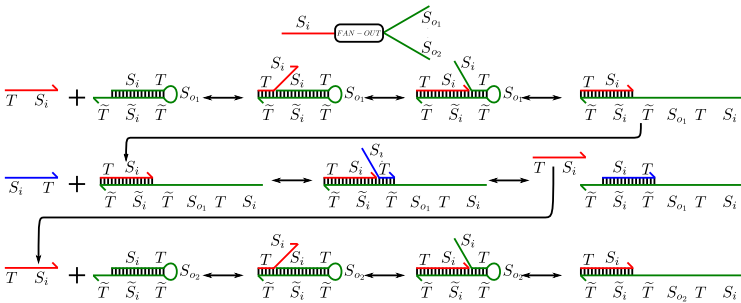
**Fig. 1.** The OR gate computing the Boolean function  $S_{i1} + S_{i2}$ . It is implemented by two green strands  $S_{i1}TS_o\tilde{T}\tilde{S}_{i1}\tilde{T}$  and  $S_{i2}TS_o\tilde{T}\tilde{S}_{i2}\tilde{T}$ . The inputs are  $TS_{i1}$  and  $TS_{i2}$ . The presence of either of these input strands triggers the exposure of the output  $TS_o$  which is initially sequestered in a hairpin.



**Fig. 2.** The AND gate computing the Boolean function  $S_{i1} \cdot S_{i2}$ . It is implemented by a complex consisting of the green strand  $S_{i2}TS_o\tilde{T}\tilde{S}_{i2}\tilde{T}$  hybridized to the light green strand  $S_{i1}T$ . The inputs are  $TS_{i1}$  and  $TS_{i2}$ . The presence of both of these input strands triggers the exposure of the output  $TS_o$  which is initially sequestered in a hairpin.



**Fig. 3.** The Propagation gate enables the signal transduction  $S_i \rightarrow S_o$ . It is implemented by a green strand  $S_iTS_o\tilde{T}\tilde{S}_i\tilde{T}$ . The input  $TS_i$  triggers the exposure of the output  $TS_o$  which is initially sequestered in a hairpin.



**Fig. 4.** A degree two fan-out gate transducing input signal  $S_i$  to two output signals  $S_{o1}$  and  $S_{o2}$

sequence  $TS_{i_2}$  is also present, it initiates strand displacement via this newly exposed toehold to reveal the output  $S_o$  as illustrated in Figure 2. Note that the reaction that opens the hairpin is irreversible in the presence of a downstream gate that consumes  $S_o$ .

Figure 3 illustrates a propagation gate, that act as a wire that propagates signal. It can be thought of as an OR gate with one of the inputs hard wired to a Boolean 0. In our implementation this is achieved by simply leaving out one of the two motifs that make up the OR gate. By stringing together a series of propagation gates, we can create signal transduction pathways between gates.

Circuits with gates that support a fan-in of 2 and a fan-out of 1 are capable of computing any boolean function. However, supporting higher fan-in and fan-out reduces the size of the circuit (number of gates) and simplifies the circuit. While unlimited fan-in and fan-out is not practical for real physical systems, we show how one may achieve a fixed small degree of fan-in and fan-out for our DNA gates. A  $k$  degree fan-in OR gate can be achieved by using  $k$  hairpin motifs in parallel. The degree of fan-in is restricted by the space available on the substrate. Overcrowding or spreading the gate over a larger area may degrade performance. The optimal arrangement would have to be experimentally determined. A  $k$  degree fan-in AND gate can be achieved by using a hairpin motif with  $k - 1$  protectors in serial. The switching speed of the multi-input AND gate is inversely propositional to the degree of fan-in due to the serial nature of the AND gate.

A  $k$  degree fan-out from a signal  $S_i$  can be implemented using  $k$  downstream propagation gates transducing the signal  $S_i$  to signals  $S_{o_1}, S_{o_2}, \dots, S_{o_k}$  using  $k - 1$  copies of a fuel strand  $S_iT$ . Fuel strands can be tethered to the substrate to achieve localized hybridization kinetics. A degree 2 fan-out gate is illustrated in figure 4. The signal  $S_i$  activates one of the  $k$  downstream propagation gates whose output region  $S_{o_i}$  is consumed by an irreversible downstream drain. The fuel strand now binds to the propagation gate using the newly exposed toehold  $\tilde{T}$  and kicks off the signal  $S_i$  which can now activate another propagation gate and so on until all the  $k$  distinct propagation gates are serially activated. Once again, transduction speed of the fanout logic is inversely propositional to the degree of fan-out due to its serial nature. This rate might be improved by tethering the  $k$  copies of the fuel strands near the propagation gates.

## 2.2 Compiling Boolean Circuits into DNA Hybridization Circuits

Converting a Boolean circuit into a DNA circuit involves two stages. First the Boolean circuit is compiled into a dual-rail circuit that computes the same function. There may be several different dual-rail circuits that compute the same function and we may wish to find an optimal one, based on characteristics like small number of gates, uniform depth, balanced signal propagation delay across all pathways (recalling that our DNA AND gates are slower than OR gates) etc. We can compile the dual-rail Boolean circuit into a DNA circuit by using the AND, OR and propagation gate motifs. We optimize the placement of these gates on our substrate, adding or deleting propagation gates as necessary.

The sequence design for the gate motifs is modular and we simply design all binding domains so as to minimize spurious interactions. Techniques for domain level sequence design have been discussed by Zhang[22].

In practice, we may incorporate further optimizations in designing our DNA circuits rather than compiling them directly. For instance, we may wish to target certain leaky portions of the circuit and make them more fault tolerant by, for instance, replicating the circuit module and taking the majority. Such techniques to increase robustness of circuits are well studied in the VLSI community and it is likely that the algorithmic solutions found there may be translated simply into DNA circuits because of the digital gate level abstraction that we provide.

### 2.3 Assembly of DNA Hybridization Circuits on Addressable DNA Nanostructures

In this section we illustrate two methods to organize DNA circuits on addressable DNA substrates by highly parallel synthesis that is experimentally feasible and scales to large number of gates.

Pistol and Dwyer[18] have demonstrated the assembly of fully addressable DNA lattices of size up to  $8 \times 8$  using hierarchical assembly techniques. The hierarchical assembly can be parallelized and has the potential to be scaled beyond  $8 \times 8$  lattices. We use this approach to organize our circuits on DNA lattices. Each DNA gate motif is designed as an extension of one of the strands that is part of the tile that assembles into a lattice. Since each tile in the finally formed lattice is uniquely addressable, the motifs can be precisely and specifically positioned on the lattice according to the circuit being implemented.

Rothemund[14] demonstrated the breakthrough DNA origami technique for manufacturing large, fully addressable DNA nanostructures with high yield in a single pot reaction. Each designed hybridization interaction in a DNA origami structure is between a staple and a region of the scaffold. No staple-staple or scaffold-scaffold interactions are designed. Thus, even if the relative concentrations of the staples are imprecise, the final yield of the origami structure remains high as long as an excess (about  $10\times$ ) of staple strands is used. We use the same approach to organize our circuits on DNA origami. Each DNA gate motif is an extension of a staple strand. Since the surface of the origami is uniquely addressable, the motifs can be precisely and specifically positioned on the origami according to the circuit being simulated.

The key cause for concern is that when annealed, the hairpin strands will interact with each other rather than folding up into the required hairpin motif. However, there is evidence that when annealed, dilute ( $\approx$  nM concentrations) interacting strands undergo uni-molecular reactions and fold up into hairpin motifs rather than hybridizing with each other via bi-molecular reactions[23]. This is explained by noting that the hairpin structure is stable at a higher temperature than the intermolecular complex and as the system is cooled, the motifs form hairpins first getting kinetically trapped in the non-optimal thermodynamic state. It is unclear if this assumption holds when the strands are locally concentrated, for instance, by tethering them close to each other. To avoid this problem,

we design our motifs such that their hairpin structure is stable at higher temperatures than both the temperature at which they are stably incorporated into the substrate and the temperature at which the bimolecular complex is stable. When annealed, we expect the hairpin motif to form while the strands are dilute and not yet tethered to the substrate, and then the motifs are incorporated into the substrate.

Since the OR motifs are simply single stranded hairpins, this is easily achieved by making the length of the specificity domain moderately longer than the length by which the motif is tethered to the origami. In practice, the tether length could be 16 bases while the length of the specificity domain could be 20 and the toehold domain could be 5 bases, making the stem of the OR hairpin motif 25 bases long. The AND motif is slightly more problematic since it is a two strand complex - a protector strand hybridized to a hairpin motif. By choosing lengths of 20 and 5 bases for the specificity and toehold domains we can ensure that the protector-hairpin complex is stable at a higher temperature than the temperature at which the origami tether is stable. However, an upstream input to the AND motif would have similar stability with the AND hairpin as the protector-hairpin complex as both are bimolecular reactions. This difficulty can be overcome in one of two ways. We can anneal the protector-hairpin complex separately, purify it and then add it to the origami mix. While annealing the origami mix we take care to not heat the sample above the melting temperature of the protector-hairpin complex. Alternately, we design the AND motif as a single hairpin motif and cleave the motif at the appropriate site after annealing by using a nicking enzyme. For this purpose we can design the toehold sequence as the recognition domain of a nicking enzyme which cleaves one of the strands of a double helix upstream of its recognition site. Note that a single nicking enzyme would be sufficient to prepare all protector-hairpin complexes and this process could be implemented in parallel. Also, the restriction enzyme won't nick the OR hairpin motifs as the corresponding position in these structures is single stranded.

## 2.4 Reusing Sequences in Spatially Separated Circuits

In the circuits we have discussed thus far, each distinct circuit element (wires and gates) is implemented by a distinct DNA sequence that is never reused. This immediately places an upper bound on the complexity of the circuits since length of DNA sequences constituting each type of circuit element is fixed. Moreover, assigning distinct sequences to different copies of the same circuit element might result in variance in their operational characteristics.

If the gates in a circuit are spatially separated into clusters such gates can interact directly only with other members of the cluster they belong to, then sequences can be reused across different clusters. Information is exchanged between clusters via signal transduction pathways. In the extreme case, one can imagine a cluster to be a single gate and that each such gate is connected to other gates via long signal transduction pathways. These pathways composed of multiple propagation gates can also benefit from domain reuse. For example the  $k$  long pathway  $W_1 \rightarrow W_2 \rightarrow \dots W_k$  can be replaced by an equivalent  $k$

long pathway  $W_a \rightarrow W_b \rightarrow W_a \rightarrow \dots W_b$ . Care should be taken when sequences are reused in signal transduction pathways running close to each other or crossing over one another. In such cases, signal flow across one pathway might initiate a spurious signal flow across the other if pathways share same sequences. Such problematic areas in the circuit can use unique sequences to minimize crosstalk.

Reusing DNA sequences to build the same functionality on a different part of the DNA hybridization circuit allows us to build complex circuits with very few distinct DNA sequences. Here we use the key property that our local DNA hybridization circuits (for example a DNA hybridization circuit for a given logical operation) are spatially separated and so cannot directly interact. While the sequence domains that take part in circuit interactions are reused, the tether sequences that position the elements on an addressable substrate can be distinct (eg. in DNA origami) or can be reused (eg. in hierarchical assembly).

## 2.5 Functional Units and Architectures

We have discussed how to build digital circuits using DNA sequences on an addressable substrate. We can build complex circuits via a hierarchical method. Small substrates can implement functional units that can be connected in a precise manner to synthesize computing architectures. The hierarchical assembly process developed by Park et al.[24] can be directly applied to build such circuits using tile based assemblies. If origami is used as a substrate, then different origami can be connected to each other via sticky ends to form larger assemblies. One could also think of using a secondary scaffold to organize different origami in a precise manner to enable information flow between them. One simple layout for such architecture would be to have the computing elements in the middle of the origami and connect them up to neighboring origami via long signal transduction pathways that terminate at the edge of the origami. An advantage of using such architectures is the ability to "plug and play" various functional units. For example, if we have designed and experimentally tested a set of functional units, say an adder, subtracter and square rooter, then we can build circuits that are composed of these functions by plugging these units into precise positions on the assembly. These functional units could be designed to ensure that they can communicate via the same signal transduction pathways for each input/output bit so that they can be composed seamlessly.

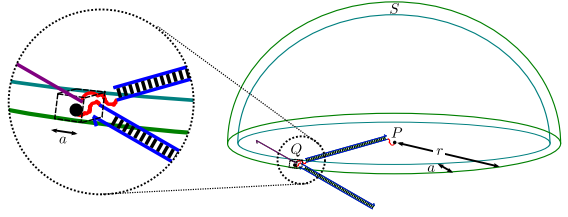
## 3 Modeling and Simulations of Tethered Systems

In this section we investigate the speedups obtained in localized hybridization circuits as follows: (i) We develop a simple biophysical model of tethered hybridization and estimate values for a toehold binding speedup factor  $\lambda$  which depends on parameters of the design of the tethers. Our biophysical model of tethered hybridization closely follows the work of Genot et al.[25] and we use data reported by Qian and Winfree[7] as a starting point for computing reaction rates. (ii) Then we use the Visual DSD system[26] to model and simulate a four bit square root circuit for various values of  $\lambda$ . We discuss some of the shortcomings of this simple model in section 4.1.

### 3.1 A Biophysical Model of Tethered Hybridization

Toehold exchange by strand displacement is modeled by Zhang and Winfree[27] as a three stage process: toehold binding, branch migration and toehold unbinding. We study the effect of tethering on a toehold exchange reaction by considering the effect of tethering on each of these stages. Branch migration and toehold unbinding rates are unlikely to be affected by tethering since they are local processes internal to the molecule. The rate and order of a toehold binding reaction changes due to tethering and essentially becomes a unimolecular reaction, taking place within a single macromolecule, with a larger rate constant. We make the biophysics motivated assumption that the speedup in the toehold binding rate constant is purely due to the effective concentration of the incoming strand. Our analysis and assumptions closely follow the work of Genot et al.[25]. We first calculate an approximate effective concentration  $c$  for the incoming toehold strand under certain biophysical assumptions. The new toehold binding rate constant  $\tilde{k}_s$  is calculated as the product of the original (diffusion-based) toehold binding rate  $k_s$  and the effective concentration  $c$  of the incoming strand.

We assume that the gates are tethered to their substrate via a short single strand of DNA that has negligible persistence length and hence acts as a completely flexible hinge. The double stranded portions of the gates are much longer than these tethers and are assumed to behave like stiff rods. Figure 5 illustrates the interaction between an upstream gate tethered at  $P$  and a downstream gate tethered at  $Q$ , a distance  $r$  away. The cube of side  $a$  illustrates the reaction volume, the volume within which the incoming toehold region must lie for toehold binding reaction to occur. We assume that  $a$  is much smaller than  $r$ . We also ignore the single stranded region at the end of the double stranded region of the upstream gate and assume that the effective concentration for toehold binding is approximately the concentration of the end of the double stranded region in the reaction volume. The probability that the end of the double stranded region lies within the reaction volume is given by:  $p = V_a/V_s$  where  $V_a = a^3$  is the reaction volume and  $V_s = 2\pi r^2 a$  is the volume of the shell  $S$  of thickness  $a$  that the end of the double stranded region can explore. Thus,  $p = a^2/2\pi r^2$ . The effective concentration in particles/m<sup>3</sup> is  $c = p/a^3 = 1/2\pi r^2 a$ . We convert this into molarity (moles per liter) by dividing by  $1000N_a$  where  $N_a = 6.023 \times 10^{23}$  is Avogadro's number. Thus  $c = 1/2000\pi r^2 a N_a$  M. The rate constant for short toehold binding in diffusion based systems is  $k_s = 5 \times 10^4$  /M/s at 25°C (see [7] supplementary information). This gives a tethered toehold rate constant of  $\tilde{k}_s = k_s \times c$  /s.



**Fig. 5.** A biophysical model of substrate tethered hybridization

The overall rate of any toehold binding reaction is the product of the scaled rate constant  $\tilde{k}_s$  and the operating concentration  $c_0$  of the assembled circuits in solution according to mass action kinetics. We approximate the kinetics of toehold binding as a bimolecular reaction and for the purposes of simulation generate a pseudo-second order rate constant scaled by the operating concentration  $c_0$ ,  $\hat{k}_s = \tilde{k}_s/c_0 = k_s \times c/c_0 = k_s \times \lambda$  where  $\lambda$  is the pseudo-bimolecular rate constant speedup. This effectively translates to a tethered toehold binding reaction rate of  $\tilde{k}_s \times c_0$ . For typical operating concentration of  $c_0 = 100$  nM,  $\lambda = 1/2000\pi r^2 N_a c_0 = 2.64 \times 10^6/ar^2$  when  $a$  and  $r$  are expressed in nanometers. The above table illustrates three values for  $\lambda$  based on different physically feasible values for  $a$  and for the length of the double stranded region  $r$ .

$r$	$a$	$\lambda$
20 nm	5 nm	$\lambda_1 = 1.32 \times 10^3 \approx 10^3$
10 nm	2 nm	$\lambda_2 = 1.32 \times 10^4 \approx 10^4$
5 nm	1 nm	$\lambda_3 = 1.06 \times 10^5 \approx 10^5$

### 3.2 Simulations of Tethered Hybridization Circuits using Visual DSD

Visual DSD [26][28] is a tool for specification, compilation and simulation of a wide class of toehold mediated DNA strand displacement reactions. Various hybridization circuits based on toehold exchange can be specified as a program in the DSD programming language along with rate constants for binding and unbinding of toeholds and for branch migrations. These programs are compiled into a set of reactions under one of four different semantics that allow modeling of the reaction at different granularities. This set of reactions can then be simulated by either a deterministic ODE solver or a stochastic Gillespie molecular simulation.

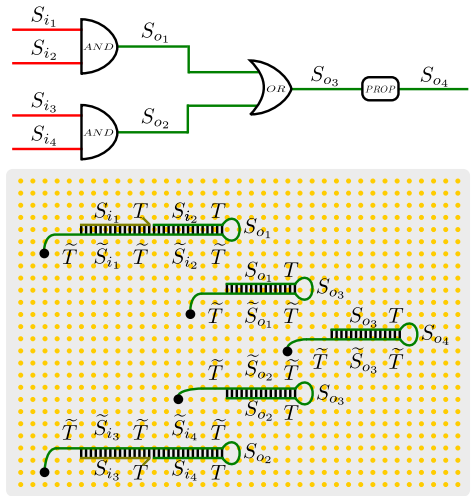
We used DSD to program a four bit square root circuit constructed out of AND, OR, FAN-OUT and PROPAGATION gates described in the previous section. As the current implementation of DSD does not allow for specification of hairpins, we modeled our systems without hairpins but enforced necessary locality by private name-spaces for different toeholds. This ensures that the output strand that physically detaches from the gate (modeling the opening of hairpins) attaches only to a very specific set of downstream gates via a private toehold. This models locality of reactions at the logical level. To model the kinetics of localized hybridization, we used rate constants reported by Qian and Winfree[7] but multiplied the toehold binding rate constant by a factor  $\lambda$  derived in the previous section. As noted earlier, localized hybridization circuits can be thought of as series of uni-molecular reactions within a macromolecule that changes state every time a reaction occurs. Our modifications to the toehold binding rate constant and its modeling as pseudo second order rate constant are consistent with this understanding. It is important to note that our simulation methodology is based on an approximation using mass action kinetics. Refer to section 4.1 for a discussion on alternate modeling approaches.

In particular, we set our toehold binding rate constant to  $\lambda \times 5 \times 10^{-5}$ /nM/s, toehold unbinding rate constant to 26/s and branch migration rate constant to 1/s. To validate the chosen rate constants, the solution based, non-localized, four bit square root circuit experimentally demonstrated by Qian and Winfree[7] was simulated with these rate constants. We also set  $\lambda = 1$  and used reported rate constants for threshold toehold binding ( $2 \times 10^{-3}$ /nM/s) and threshold toehold unbinding (1.3/s). DSD simulations with these rate constants agreed well with experimental data reported by Qian and Winfree[7]<sup>1</sup>.

As an initial test we implemented the circuit illustrated in figure 6 in DSD using our simulation methodology<sup>1</sup>. The reaction graph for this system is shown in figure 7. In addition to using private toeholds to model locality as described previously, we also used a common toehold with unmodified binding rate constant for the input strands, to model the diffusion of inputs to the substrate.

Figure 8a shows the stochastic simulation of the circuit compiled with finite semantics with unproductive reactions switched on, for different values of  $\lambda$ . Finite semantics mode in DSD models the toehold exchange reactions as a three step process of toehold binding, branch migration and toehold unbinding and assigns specific rates to each process. Unproductive reactions occur when branch migration does not follow after successful toehold binding. This happens when the specificity domain of the incoming strand is different from the specificity domain of the gate which is caused when multiple specificity domains share the same toehold. For more details about the Visual DSD tool and the options it offers see [28].

The simulations show marked improvement in overall reaction rates with increasing  $\lambda$  but diminishing gains as  $\lambda$  increases beyond 1000. This can be attributed to the fact that the overall reaction kinetics is governed by the rate limiting step of input diffusion at higher  $\lambda$ .

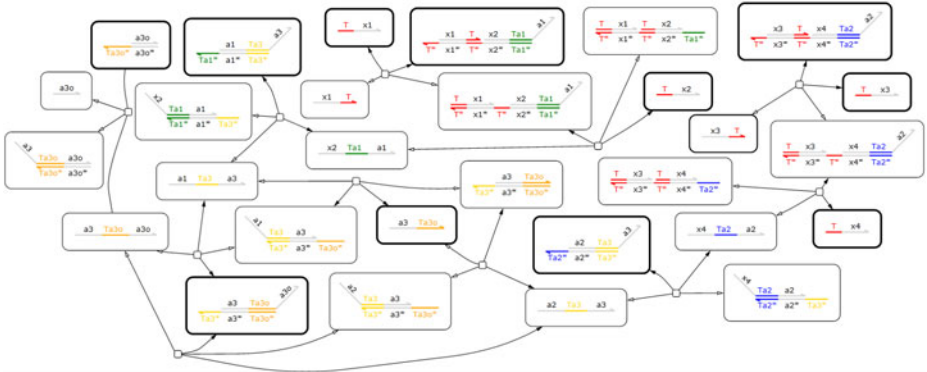


**Fig. 6.** Example computation of  $(S_{i1} \cdot S_{i2} + S_{i3} \cdot S_{i4}) \rightarrow S_{o4}$  implemented by two AND gates, an OR gate and a Propagation gate indicated by green strands and complexes. The addressable substrate is illustrated by the gray rectangle and the points of addressability are illustrated by the yellow dots. The gates are placed at specific locations, indicated by big black dots, such that down stream gates are next to their corresponding inputs from the upstream gates.

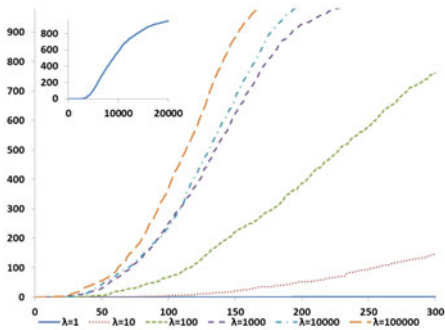
<sup>1</sup> Data and code available at

<http://research.microsoft.com/dna/dna17localized.zip>

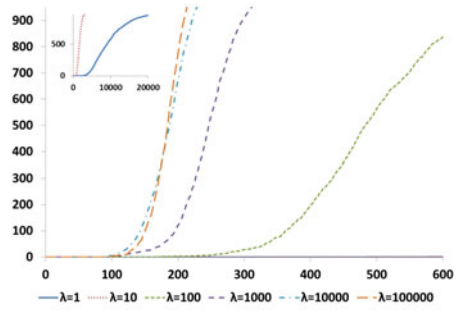




**Fig. 7.** Reaction graph for the example circuit illustrated in figure 6 implementing the Boolean function  $x_1 \cdot x_2 + x_3 \cdot x_4$ . Initial species are highlighted with darker borders. Note that the input strands  $Tx1, Tx2, Tx3, Tx4$  have a common universal toehold  $T$  whose binding rate constant is set to  $5 \times 10^{-5}$  /nM/s while the other toeholds are private (for example  $Ta2$  is private to domain  $a2$ ) and their binding rate constants are set to  $\lambda \times 5 \times 10^{-5}$  /nM/s.



**(a)** Simulation data for circuit implementing  $x_1 \cdot x_2 + x_3 \cdot x_4$  with inputs set to  $x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 0$  for different  $\lambda$ . Simulation time: 300 seconds. Simulation carried out in finite mode with unproductive options on. Inset shows longer run of system for  $\lambda = 1$ .



**(b)** Simulation data for output LSB from the square root circuit with input 1100 for different  $\lambda$ . Simulation time: 600 seconds. Simulation carried out in finite mode with unproductive options on. Inset shows longer run of system for  $\lambda = 1, 10$ .

**Fig. 8.** Simulation data

Encouraged by these results we programmed a four bit square root circuit using AND, OR and FAN-OUT gates described earlier. As before, we used private toeholds with modified binding rate constants to model locality and used a universal toehold with standard rate constants to model diffusion of inputs. Figure 8b shows data from simulations using DSD in finite mode with unproductive reactions switched on. We first simulated the four bit square root circuit for all

input values with  $\lambda = 1$  and found that the computation of the output LSB for input 1100 was the slowest to 95% completion. Due to space consideration and in the spirit of worst case analysis, we show the behavior of this signal at different values of  $\lambda$ . More data from our simulations is available at this URL<sup>1</sup>. As before, completion rates show dramatic improvements with increasing  $\lambda$  and the overall kinetics is rate limited at higher  $\lambda$  by input diffusion (which roughly takes 150 seconds at 100 nM).

## 4 Discussion

### 4.1 Refined Modeling and Simulations of Tethered Systems

All localized toehold binding interactions are assigned the same rate constant in our simulations. However, there are two types of localized toehold binding, one between regions on the same strand (resulting in hairpin formation) and another between distinct strands. Our biophysical model only applies to the latter case. A more careful analysis would include different binding rates in both cases. The challenge of modeling and simulation of tethered systems is to model uni-molecular reactions rather than the bimolecular reactions found in most other conventional hybridizations reaction systems. Our biophysical model is preliminary and ignores the worm-like chain behavior of single strands of DNA. A more detailed model may give a better estimation of local concentrations.

Molecular circuits tend to leak: downstream hybridization cascades are sometimes set off by thermal noise even in the absence of upstream signals. Spurious hybridization interactions, either due to unintended sequence complementarity or unproductive interactions between complementary domains (e.g. universal toehold binding), is another common problem. Sequence reuse can mitigate unintended sequence complementarity while careful positioning of interfering strands on the substrate can inhibit unproductive reactions. Nevertheless we expect leaks and unproductive reactions to exist, and hence the need for a model of leak reactions in localized circuits. A fruitful approach may be to model the time to failure of each gate as a random variable and use this to estimate the overall leak rate. We discuss some error-tolerance mechanisms in section 4.4.

We simulate tethered systems using mass action kinetics. An alternate approach would be to model the entire network as a continuous time Markov chain and estimate (or simulate) its expected time to completion where the times to completion of toehold binding, branch migration and toehold unbinding could be estimated via DNA molecular dynamics. Our preliminary simulations (data not shown) under this regime indicates higher speedups in tethered systems. Addressing these issues is beyond the scope of this paper and will be a part of future work in tethered hybridization systems.

### 4.2 Optimizations

The circuits discussed thus far have only utilized one side of the addressable surface. Suppose the substrate is addressable on both sides and the substrate is

stiff and dense enough to ensure that strands on one side cannot interact with the strands on the other (note that these assumptions are true for certain DNA origami). Then it is possible to use both sides of the substrate for implementing different circuits. These circuits can still interact at the edges of the substrate for signal transduction.

The output of simple circuits could be a Boolean value requiring just one bit. The output of such computation can be detected via standard fluorophore/quencher protocols. But complex circuits might output multiple bits or compute an integer requiring the detection of multiple bits. Though one can use multiple fluorophore/quencher pairs operating at various frequencies, it is quite clear that this solution does not scale. One possible way to overcome this issue is to implement the circuit for each bit of the output in different test tubes. Once again, we can exploit the spatial separation of circuits to reuse the same fluorophore/quencher pair for every output bit.

To assemble large circuits on origami requires longer scaffolds. Naturally occurring long scaffolds might be problematic since they might exhibit strong secondary structure or might interfere with the DNA sequences used for various gates. If we use spatial separation to restrict ourselves to a small set of DNA sequences, we might mitigate the latter problem. The former problem might be solved by careful design of a synthetic scaffold.

### 4.3 Synchronous Computation and Nanomanufacture

The circuits described in this paper were asynchronous and used dual rail logic. It is possible to achieve synchronous lock-step computation using AND gates. For example, if we want one part of a circuit (say part B) to be activated only after another part of the circuit (say part A) has finished its computation, then each signal transduction pathway entering into B can be changed into the output of the AND of that original pathway and a specific completion signal from part A. Thus part B is locked unless part A is complete. Alternating this strategy across two circuits allows them to proceed in a lock-step fashion.

This technique can be extended to nanomanufacture applications. We can think of the entire process having two components, a fabricating nanomachine like a DNA walker and a computing logic that governs the action of the fabricating device. The fabricating device and the computing logic can be operated in the lock step fashion described earlier. The computation leading to this product formation can be governed by the actual inputs to the circuit.

### 4.4 Possible Errors and Techniques to Mitigate Them

Errors with localized DNA circuits can broadly be classified into two types: errors in organizing the gate motifs on the substrate and errors in operation. The chief possible errors in organizing the motifs on the substrate are: (i) missing motifs and (ii) damaged motifs due to incorrect folding, sequence truncation or formation of spurious bimolecular complexes. Techniques to deal with these kinds of errors are discussed in section 2.3.

Errors in operation are chiefly due to: (i) leaks via spontaneous opening of the hairpin motifs (ii) leaks via stacking induced strand displacement and (iii) spurious toehold binding. Spontaneous opening of the hairpin motifs are likely to be rare at our operation conditions, since the stem of the hairpin is 25 bases. We refer to the end of the stem at the loop region as the head of the motif and the other end as its tail. Stacking induced strand displacement is likely to occur via head to tail stacking of motifs. The loop region is likely to sterically hinder such stacking, destabilizing it. We will also experiment with carefully orienting the motifs on the origami surface such that these stackings strain the motif tether region and are hence sterically hindered. For instance the motifs likely to undergo stacking could be oriented alongside each other. Since each motif has the same toehold binding region, the output of one motif may bind to the toehold region of a neighboring motif even if they are not designed to interact. This spurious interaction is prevented from setting off downstream reactions by the mismatch in the specificity domains. However, such reactions may block the toehold region and slow down the operation of the circuit. This problem is present even with the seesaw circuit of Qian and Winfree[7] but does not seem to significantly affect their correct operation for moderate circuit sizes. The spurious toehold interactions in our designs are restricted to the diameter of motifs reachable by the tethered motif, in contrast to the seesaw circuits where it is a global problem.

In spite of these techniques, a certain level of leaks is unavoidable. If we assume that every copy of the circuit on origami has a fixed independent failure probability of  $\epsilon$ , then we expect that out of  $N$  targeted copies of the circuit,  $N(1 - \epsilon)$  of them will function correctly. The final output of the circuit is the consensus of the outputs across all copies of the circuit, with appropriately set thresholds based on the failure rate. Alternately, we can implement standard techniques in fault tolerance into our circuits. This correction of errors at the logical level has been used with great success in building of semiconductor based circuits and this provides inspiration in dealing with errors due to leaks.

## 5 Conclusions

Local bimolecular reactions have multiple advantages over global molecular computation. In this paper we have developed detailed designs to implement DNA circuits on fully addressable DNA nanostructures such as a fully addressable lattice developed by Yan et al.[13] or DNA origami developed by Rothemund[14]. In doing so we developed a local molecular computing methodology to compute arbitrary Boolean functions. Our circuits are designed carefully to place downstream gates close enough to upstream gates to implement rapid signal transduction but far enough to minimize leaks. We argued that our circuits will: (i) be faster than chemical reaction networks due to increased local concentration of reacting species, (ii) exhibit generally sharper switching behavior and higher precision due to single molecule interactions, (iii) be highly parallel since each circuit operates independently of the others which finds use in nanomanufacture

(iv) be modular and scalable due to ability to reuse DNA sequences in spatially separated regions. A biophysical model of localized hybridization reactions was used to estimate the effect of locality on reaction rates. We used the Visual DSD simulation software in conjunction with these localized reaction rates to simulate a localized circuit for computing the square root of a four bit number.

This effort is a first attempt at realizing enzyme-free localized hybridization circuits. In light of the rapid growth of DNA nanotechnology, it is our hope that the principles expounded in this paper will serve as a starting point for the eventual realization of localized hybridization circuits in the laboratory.

**Acknowledgments.** We wish thank to Erik Winfree, David Zhang and Bernard Yurke for useful discussions on the localized strand displacement process. We thank anonymous referees for pointing out relevant prior work and critical suggestions on modeling of tethered systems. We would also like to thank Sudhanshu Garg for assisting in creating figures for a preliminary draft and Archana Ramamoorthy for proof-reading. This work was supported by NSF EMT Grants CCF-0829797 and CCF-0829798.

## References

1. Adleman, L.: Molecular Computation of Solutions to Combinatorial Problems. *Science* 266(5178), 1021–1024 (1994)
2. Sherman, W., Seeman, N.: A Precisely Controlled DNA Biped Walking Device. *Nano Letters* 4, 1203–1207 (2004)
3. Zhang, D., Turberfield, A., Yurke, B., Winfree, E.: Engineering Entropy-Driven Reactions and Networks Catalyzed by DNA. *Science* 318, 1121–1125 (2007)
4. Yin, P., Choi, H., Calvert, C., Pierce, N.: Programming Biomolecular Self-assembly Pathways. *Nature* 451(7176), 318–322 (2008)
5. Dirks, R., Pierce, N.: Triggered Amplification by Hybridization Chain Reaction. *Proceedings of the National Academy of Sciences of the United States of America* 101(43), 15275–15278 (2004)
6. Sakamoto, K., Kiga, D., Momiya, K., Gouzu, H., Yokoyama, S., Ikeda, S., Sugiyama, H., Hagiya, M.: State Transitions by Molecules. *Biosystems*, 81–91 (1999)
7. Qian, L., Winfree, E.: Scaling up Digital Circuit Computation with DNA Strand Displacement Cascades. *Science* 332(6034), 1196–1201 (2011)
8. Rothmund, P., Winfree, E.: The Program-Size Complexity of Self-Assembled Squares. In: *Symposium on Theory of Computing*, pp. 459–468 (2000)
9. Turberfield, A., Mitchell, J., Yurke, B., Mills, A., Blakey, M., Simmel, F.: DNA Fuel for Free-Running Nanomachines. *Physical Review Letters* 90(11) (2003)
10. Seelig, G., Yurke, B., Winfree, E.: Catalyzed Relaxation of a Metastable DNA Fuel. *Journal of the American Chemical Society* 128(37), 12211–12220 (2006)
11. He, Y., Liu, D.: Autonomous Multistep Organic Synthesis in a Single Isothermal Solution Mediated by a DNA Walker. *Nature Nanotechnology* 5(11), 778–782 (2010)
12. Gu, H., Chao, J., Xiao, S.-J., Seeman, N.: A Proximity-based Programmable DNA Nanoscale Assembly Line. *Nature* 465(7295), 202–205 (2010)

13. Yan, H., Park, S.H., Finkelstein, G., Reif, J., LaBean, T.: DNA-Templated Self-Assembly of Protein Arrays and Highly Conductive Nanowires. *Science* 301(5641), 1882–1884 (2003)
14. Rothemund, P.: Folding DNA to Create Nanoscale Shapes and Patterns. *Nature* 440, 297–302 (2006)
15. Qian, L., Winfree, E.: A Simple DNA Gate Motif for Synthesizing Large-scale Circuits. *DNA Computing*, 70–89 (2009)
16. Cardelli, L.: Two-Domain DNA Strand Displacement. *DCM*, 47–61 (2010)
17. Park, S.-H., Yin, P., Liu, Y., Reif, J., LaBean, T., Yan, H.: Programmable DNA Self-assemblies for Nanoscale Organization of Ligands and Proteins. *Nano Letters* 5, 729–733 (2005)
18. Pistol, C., Dwyer, C.: Scalable, Low-cost, Hierarchical Assembly of Programmable DNA Nanostructures. *Nanotechnology* 18, 125305–125309 (2007)
19. Lin, C., Liu, Y., Yan, H.: Self-Assembled Combinatorial Encoding Nanoarrays for Multiplexed Biosensing. *Nano Letters* 7(2), 507–512 (2007)
20. Douglas, S., Dietz, H., Liedl, T., Hogberg, B., Graf, F., Shih, W.: Self-assembly of DNA into Nanoscale Three-dimensional Shapes. *Nature* 459(7245), 414–418 (2009)
21. Dietz, H., Douglas, S., Shih, W.: Folding DNA into Twisted and Curved Nanoscale Shapes. *Science* 325(5941), 725–730 (2009)
22. Zhang, D.: Towards Domain-Based Sequence Design for DNA Strand Displacement Reactions. *DNA* 16, 162–175 (2010)
23. Dirks, R., Bois, J., Schaeffer, J., Winfree, E., Pierce, N.: Thermodynamic Analysis of Interacting Nucleic Acid Strands. *SIAM Review* 49, 65–88 (2007)
24. Park, S.H., Pistol, C., Ahn, S.J., Reif, J., Lebeck, A., LaBean, C.D.T.: Finite-Size, Fully Addressable DNA Tile Lattices Formed by Hierarchical Assembly Procedures. *Angewandte Chemie International Edition* 45(5), 735–739 (2006)
25. Genot, A., Zhang, D., Bath, J., Turberfield, A.: Remote Toehold: A Mechanism for Flexible Control of DNA Hybridization Kinetics. *Journal of American Chemical Society* 133(7), 2177–2182 (2011)
26. Phillips, A., Cardelli, L.: A Programming Language for Composable DNA Circuits. *Journal of The Royal Society Interface* 6(11), 419–436 (2009)
27. Zhang, D.Y., Winfree, E.: Control of DNA Strand Displacement Kinetics Using Toehold Exchange. *Journal of the American Chemical Society* 131(48), 17303–17314 (2009)
28. Lakin, M., Youssef, S., Cardelli, L., Phillips, A.: Abstractions for DNA Circuit Design. *Journal of The Royal Society Interface* (in press, 2011)

# Less Haste, Less Waste: On Recycling and Its Limits in Strand Displacement Systems

Anne Condon, Alan Hu, Ján Maňuch, and Chris Thachuk

The Department of Computer Science,  
University of British Columbia, Vancouver, British Columbia, V6T 1Z4

**Abstract.** We study the potential for molecule recycling in chemical reaction systems and their DNA strand displacement realizations. Recycling happens when a product of one reaction is a reactant in a later reaction. Recycling has the benefits of reducing consumption, or waste, of molecules and of avoiding fuel depletion. We present a binary counter that recycles molecules efficiently while incurring just a moderate slowdown compared to alternative counters that do not recycle strands. This counter is an  $n$ -bit binary reflecting Gray code counter that advances through  $2^n$  states. In the strand displacement realization of this counter, the waste—total number of nucleotides of the DNA strands consumed—is  $O(n^3)$ , while alternative counters have  $\Omega(2^n)$  waste. We also show that our  $n$ -bit counter fails to work correctly when  $\Theta(n)$  copies of the species that represent the state (bits) of the counter are present initially. The proof applies more generally to show that a class of chemical reaction systems, in which all but one reactant of each reaction are catalysts, are not capable of computations longer than  $\frac{1}{2}n^2$  steps when there are at least  $n$  copies.

## 1 Introduction

DNA strand displacement systems support simulation of logic circuits and DNA walkers, and can in principle support general purpose computation in an energy-efficient manner [4, 7, 8, 10, 15–18, 20]. The computations typically consume strands at all reaction steps. Catalyst strands are an exception in that they are not consumed during the course of a reaction, but are recycled to perform the same operation multiple times.

Can strand displacement systems recycle strands in more general ways? We show that the answer is yes: we describe chemical reaction system computations, and their strand displacement realizations, where recycling of strands significantly reduces waste and avoids fuel depletion while incurring just a moderate slowdown relative to comparable computations that do not recycle strands. Thus our title: less haste, less waste. Our recycling computations are binary counters—simple and yet fundamental constructs in computation. A new feature of our strand displacement constructions is a mutex synchronization primitive, which ensures that reactions proceed atomically in the sense that all products of one reaction have been released before the next starts. The second contribution of the paper is to demonstrate a limit to recycling: recycling is not possible in certain classes of strand displacement systems that should work correctly even when many copies of the initial state of the system are present in the same environment.

The rest of this introduction illustrates the concept of strand recycling and gives an overview of our results and related work. Sections 2 and 3 then provide the technical details of the strand-recycling counters and the limits of recycling.

### 1.1 On the Potential for Strand Recycling

We illustrate the concept of recycling using a 3-bit counter that is specified as a chemical reaction system—details of a strand displacement implementation are in Section 2. The counter follows the sequence of bit values shown in the left column of Fig. 1(a). The counter is a binary reflecting Gray code counter [13]. A Gray code counter advances in such a way that exactly one bit changes at each step. A binary reflecting Gray code counter gets its name from the following property: if the states of the counter are written in a column starting from  $0_n 0_{n-1} \dots 0_1$  and a line is drawn just below row  $2^{i-1}$ , where bit  $i$  changes from  $0_i$  to  $1_i$ , then in the next  $2^{i-1}$  rows the values of the low order  $i - 1$  bits are the reflection of those above the line. For example, consider bits  $b_2$  and  $b_1$  of the 3-bit sequence in Fig. 1(a): the last four rows are a reflection of the first four rows. We call the resulting sequence of states the *Gray code sequence*.

Fig. 1(b) gives the chemical reaction system for this counter, which we call GRAY. The state of the 3-bit GRAY counter is determined by three *signal* molecules, one per bit. Presence of a single copy of signal  $0_i$  denotes that the  $i^{\text{th}}$  bit has value 0 while presence of a single copy of  $1_i$  denotes that the  $i^{\text{th}}$  bit has value 1. The initial counter state is  $0_3 0_2 0_1$  and the reactions ensure that exactly one of  $0_i$  and  $1_i$  is present at any time. The counter advances through application of the three reversible chemical reactions (1-3) of Fig. 1(b). Each row of the table in Fig. 1(a) lists the reaction needed to produce the subsequent row; for example, the counter advances from  $0_3 0_2 1_1$  to  $0_3 1_2 1_1$  via reaction (2) in the forward direction (2-for).

In realizing these reactions with strand displacement systems (see Section 2), additional strands that are not shown in the chemical reaction system are consumed and produced; we call these additional strands *transformers*. For example, transformers might serve to ensure that all reactants are available before any product is produced, or may be side-products of a reaction that have no further use. Suppose that a set of strands  $T_i^f$  is consumed and a set  $T_i^r$  is produced when reaction ( $i$ ) takes place in the forward direction; conversely  $T_i^r$  is consumed and  $T_i^f$  is produced when reaction ( $i$ ) takes place in the reverse direction.

The key point is that in most of the rows, the signal molecules and transformer strands that are consumed were produced by reactions of earlier rows and are thus recycled. For example, in the third step the counter advances from state  $0_3 1_2 1_1$  to  $0_3 1_2 0_1$ , uses reaction (1) in the reverse direction (1-rev) and consumes the signal molecule  $0_1$  and the set of transformer strands  $T_1^r$  that were produced in the first step, thereby recycling  $T_1^r$ . Only in the three rows  $0_3 0_2 0_1$ ,  $1_3 1_2 0_1$  and  $0_3 1_2 0_1$ —precisely those rows when a reaction occurs for the first time—the molecules consumed are not produced in earlier rows. In contrast, a chemical reaction system for a standard binary counter produces waste molecules at every step and these waste molecules are never recycled in subsequent steps.

Recycling in DNA strand displacement systems offers the potential of supporting energy-efficient DNA computations in which the waste, or number of strands consumed,



$b_3$	$b_2$	$b_1$	Reaction	Consumed	Produced
0 <sub>3</sub>	0 <sub>2</sub>	0 <sub>1</sub>	(1-for)	$\mathcal{T}_1^f$	$\mathcal{T}_1^r$
0 <sub>3</sub>	0 <sub>2</sub>	1 <sub>1</sub>	(2-for)	$\mathcal{T}_2^f$	$\mathcal{T}_2^r$
0 <sub>3</sub>	1 <sub>2</sub>	1 <sub>1</sub>	(1-rev)	$\mathcal{T}_1^r$	$\mathcal{T}_1^f$
0 <sub>3</sub>	1 <sub>2</sub>	0 <sub>1</sub>	(3-for)	$\mathcal{T}_3^f$	$\mathcal{T}_3^r$
1 <sub>3</sub>	1 <sub>2</sub>	0 <sub>1</sub>	(1-for)	$\mathcal{T}_1^f$	$\mathcal{T}_1^r$
1 <sub>3</sub>	1 <sub>2</sub>	1 <sub>1</sub>	(2-rev)	$\mathcal{T}_2^r$	$\mathcal{T}_2^f$
1 <sub>3</sub>	0 <sub>2</sub>	1 <sub>1</sub>	(1-rev)	$\mathcal{T}_1^r$	$\mathcal{T}_1^f$
1 <sub>3</sub>	0 <sub>2</sub>	0 <sub>1</sub>			

(b)

(1)  $\mathbf{0}_1 \rightleftharpoons \mathbf{1}_1$

(2)  $\mathbf{0}_2 + \mathbf{1}_1 \rightleftharpoons \mathbf{1}_2 + \mathbf{1}_1$

(3)  $\mathbf{0}_3 + \mathbf{1}_2 + \mathbf{0}_1 \rightleftharpoons \mathbf{1}_3 + \mathbf{1}_2 + \mathbf{0}_1$

**Fig. 1.** (a) Enumeration of counter states (left three columns), reaction that advances the state from one row to the next and its direction—forward (for) or reverse (rev)—and sets of transformer molecules consumed and produced. (b) Chemical reaction system for a three-bit binary reflecting Gray code counter. Species that appear on just one side of the reaction are shown in boldface and correspond to the bit that changes value as a result of the reaction. To ensure correctness, additional “catalyst” species appear on both sides and the corresponding bits are unchanged. At any step, only one reaction is applicable to advance the counter, although since the reactions are reversible the counter could also retreat to its previous value.

is logarithmic in the length of the computation. Systems that recycle strands do not use fuel, *i.e.*, large concentrations of certain transformer species that bias reactions in one direction, and so are not prone to problems of fuel depletion or fuel leakage. However, such advantages come at a price: our counter proceeds somewhat more slowly—is less hasty—than comparable fuel-driven strand displacement counters. The slowdown is due in part to the fact that reactions are used in both directions. Thus, our GRAY counter is not biased to advance towards the final state but rather performs an unbiased random walk, both advancing and retreating, ultimately reaching the final state. We also describe a counter, GRAY-FO that uses reactions in which the reactions are of fixed order, *i.e.*, the maximum number of reactants and products in any reaction are fixed, independent of the number of counter bits.

Table 1 summarizes properties of our counters and compares with another counter, which we call QSW, based on work of Qian *et al.* [9] (see Section 1.3). The properties considered are (i) *order* or max number of reactants or products of chemical reactions that describe the counter, (ii) *waste* or total number of nucleotides needed to implement the counter and (iii) *haste* or expected time for the counter to reach a designated final state from its initial state, when the volume equals the waste. Our  $n$ -bit binary reflecting Gray code counter, GRAY, uses reactions of order  $\Theta(n)$ , generates only  $\Theta(n^3)$  waste and uses expected time  $\Theta(n^3 2^{2n})$  to reach the final state. Our GRAY-FO counter improves on the GRAY counter in that the reaction order is  $\Theta(1)$ . The QSW counter also has reaction order  $\Theta(1)$  and has expected time  $\Theta(2^{2n})$ , which is somewhat better than the expected time needed by our counters. However, the QSW counter generates  $\Theta(2^n)$  waste, exponentially worse than our counters. All three counters are deterministic in that they advance and retreat through a predetermined linear ordering of states.

**Table 1.** Comparison of  $n$ -bit counter implementations. The GRAY and GRAY-FO counters are described in Section 2. The QSW counter is based on the simulation of stack machines by strand displacement reactions of Qian *et al.* [9].

Properties	GRAY	GRAY-FO	QSW [9]
Reaction order	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$
Consumption (waste)	$\Theta(n^3)$	$\Theta(n^3)$	$\Theta(2^n)$
Exp. Time (haste)	$\Theta(n^3 2^{2n})$	$\Theta(n^3 2^{2n})$	$\Theta(2^{2n})$

## 1.2 On the Limits of Strand Recycling

The proof that our  $n$ -bit GRAY counter advances correctly through  $2^n$  states assumes that only single copies of initial species are present. In Section 3, we show that if  $\Theta(n)$  copies of the initial species are present, then the counter does not advance properly in a very strong sense: the final state of the counter can be reached in just  $O(n^2)$  chemical reactions, rather than using the intended sequence of  $2^n$  reactions. More generally, under some restrictions on the allowable chemical reaction systems, we prove that in any realization of the system with  $\Omega(|S|)$  copies of the initial species, any species of the system can be generated in  $O(|S|^2)$  reaction steps, where  $S$  is the number of distinct species. In particular, if the waste of such a chemical reaction system is logarithmic in the length of a valid computation, the system does not work correctly when many copies of the initial reactants are present.

## 1.3 Related Work

In related work, Qian *et al.* [9] showed how to simulate a stack machine using strand displacement systems. A binary counter can be implemented via a stack machine; we call such a counter a QSW (Qian-Soloviechik-Winfree) counter and we compare its properties and resources with our counters in Table 1. Details are provided in Section 2.6.

Cardelli [1, 6] has shown how primitives that support concurrent models of computation, such as fork and join gates, can be implemented using strand displacement systems. Many of our techniques are similar to those of Cardelli’s constructions: for example, our signal strands share a common toehold while the long domains are distinct, and we do not use branched structures. To effect an abstract chemical reaction with  $i$  reactants and  $i$  products, we use cascading of strand exchanges whereby the reactants are first absorbed (by transformer molecules) and products are then released by further strand exchanges. This order of events is similar to an  $i$ -way join followed by an  $i$ -way fork of Cardelli; it is similar also to the strand displacement realizations of  $i$ -way molecular reactions of Qian *et al.* [9]. A new feature of our constructions is the use of a *mutex* strand to ensure that the  $(k + 1)^{\text{th}}$  reaction of a deterministic computation does not proceed until all products of the  $k^{\text{th}}$  reaction have been produced.

Building on models of Winfree and Rothemund [12, 19], Reif *et al.* [11] studied a tile-based graph assembly model in which tiles may both adhere to and be removed from a tile assembly. In their self-destructible graph assembly model, the removal of tiles allows for the possibility of tile reuse. The authors demonstrate that tile reuse is possible in an abstract tile model, via a PSPACE-hardness result. Doty *et al.* [2] showed a negative result on tile reuse for an irreversible variant of the model of Reif *et al.*

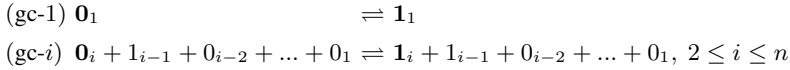
Kharam *et al.* [5] describe a DNA binary counter in which bit values are represented using relative concentrations of pairs of molecules. This is very different than our work in this paper, where the values of bits (0 and 1) are represented by the absence or presence of certain signal molecules.

## 2 GRAY: A Binary Reflecting Gray Code Counter

Here we describe the chemical reaction system and strand displacement implementation of our GRAY counter, provide a proof of its correctness, and analyze its expected time (haste) and space usage (waste). We show how it can be modified to use only bi-molecular reactions, resulting in our fixed-order GRAY counter: GRAY-FO. We also compare our counters to the QSW counter.

### 2.1 Chemical Reaction System for the GRAY Counter

We generalize the 3-bit GRAY counter in Section 1.1 in the obvious manner to  $n$ -bits. The counter state is represented by  $n$  signal molecules, one per bit. Presence of signal molecule  $b_i$  denotes that the  $i^{\text{th}}$  bit has value  $b_i$ , for  $b = 0$  or  $b = 1$ . Initially, the state is  $0_n \dots 0_2 0_1$ . Each possible state of the counter represents a value in the Gray code sequence. The counter is described abstractly by the following chemical reactions:



**Lemma 1.** *The above chemical reaction system ensures the GRAY counter, when in state  $v$ , can only advance to state  $v_{\text{next}}$ , or retreat to state  $v_{\text{prev}}$ , corresponding to the next or previous value in the Gray code sequence, respectively, if each reaction is atomic, and all initial signal molecules exist as single copies.*

*Proof.* First, observe that at any state of the system, for each  $i$ , exactly one of the signals  $0_i$  and  $1_i$  is present (in an unbound state). Hence, at any state of the system only two reactions can be applied: (gc-1) and (gc- $i$ ), where  $i$  is the smallest index such that signal  $1_{i-1}$  is present. Indeed, the reactions (gc- $j$ ), where  $2 \leq j < i$ , cannot be applied as, by the definition of  $i$ , signal  $0_{j-1}$  is present, and hence,  $1_{j-1}$  is not present. Similarly, the reactions (gc- $j$ ), where  $j > i$ , cannot be applied as signal  $1_{i-1}$  is present, and hence,  $0_{i-1}$  is not present. It follows that at any state of the system, the system can only progress in the forward or the backward directions.  $\square$

### 2.2 Strand Displacement Implementation of the GRAY Counter

A strand displacement implementation of the GRAY counter requires a means to simulate the chemical reaction equations. Furthermore, the correctness of the counter is predicated on the assumption that each chemical reaction is *atomic*. Qian *et al.* [9] proposed a construction—hereafter called the QSW construction—that is capable of simulating bi-molecular, and higher-order, chemical reactions. Specifically, the construction can exchange a set of signals (the reactants) for another set of signals (the products)

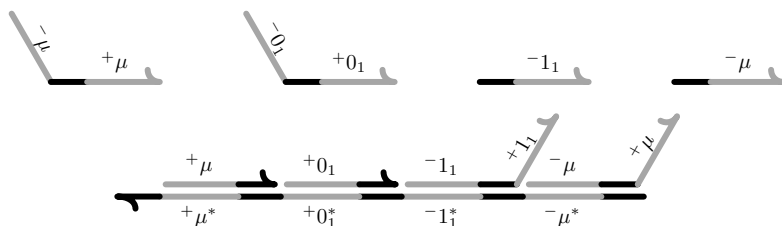
through a sequence of strand displacement events. Unfortunately, the construction is not atomic, since some product signals can start initiating other reactions before all product signals are produced. However, the strand displacements do occur in a fixed order and all reactant signals are consumed before any product signal is produced. We exploit this fact to simulate atomicity.

In particular, we borrow the concept of *transactions* from digital computation—a group of operations either completes or does not complete in its entirety. We achieve this with the use of a simple synchronization primitive: a *mutex*. The state of our counter is only defined when the mutex is available. This is analogous to processes blocking when attempting to read a memory location currently being written to by another. Let  $\mu$  denote a single copy of a signal molecule representing the mutex. In any sequence of strand displacements representing a chemical reaction,  $\mu$  is the first reactant to be consumed and the last product to be produced. Therefore only one reaction (transaction) can be in progress at any given time. When  $\mu$  is next available, either all strand displacements in the sequence took place and the counter is in a new state—the transaction succeeded—or the counter is in the same state and the configuration of all molecules is exactly the same prior to the reaction beginning—the transaction failed. Since each reaction is implemented as a transaction, they *appear atomic*.

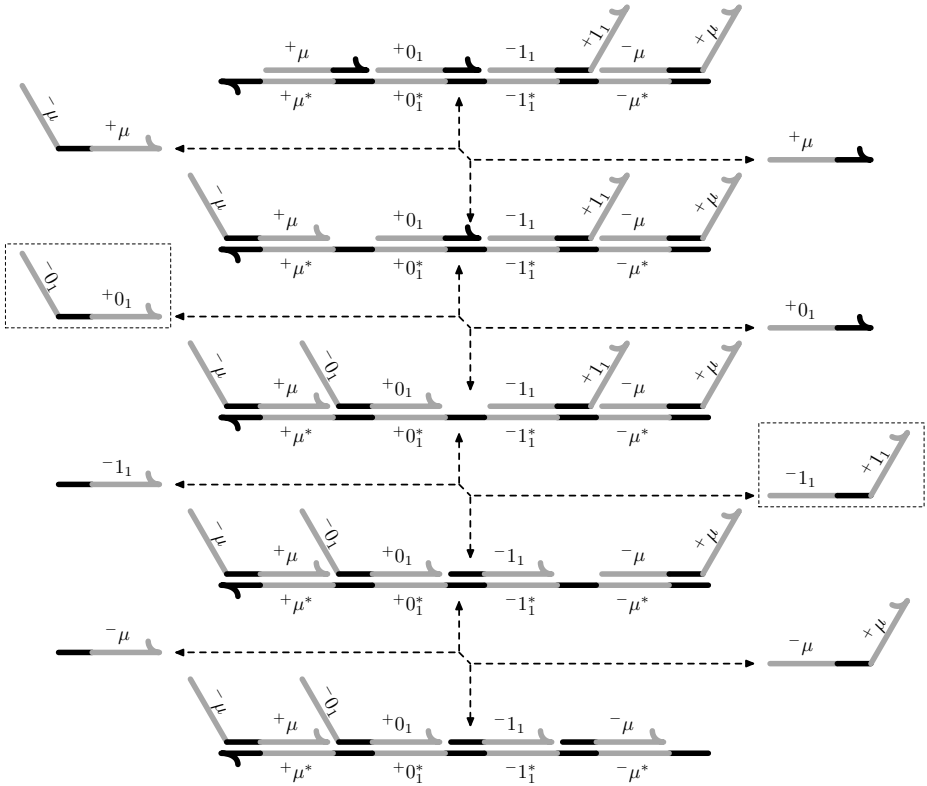
We will use only one type of toehold, and therefore we will not label toeholds in the figures below. All signals in the QSW construction are of the same form: a negative recognition domain  $-d$ , followed by a toehold  $t$ , followed by a positive recognition domain  $+d$ . The construction also uses auxiliary strands consisting of a single domain and a single toehold, and one (saturated) template strand initially bound to signal and auxiliary strands. We refer to the saturated template complex and associated auxiliary strands, collectively, as a transformer. An example of the signal molecules and the transformer associated with the forward direction of the reaction  $0_1 \rightleftharpoons 1_1$  is given in Fig. 2.

As previously discussed, the reaction can only initiate if the signal molecule  $\mu$  is present, and can only complete if all other reactants—in this case  $0_1$ , assuming a forward reaction—are available. An example of the sequence of strand displacements for the reaction  $0_1 \rightleftharpoons 1_1$  is given in Fig. 3. The reaction proceeds from top to bottom in the forward direction and from bottom to top in the backwards direction.

The transformers that implement the  $i^{\text{th}}$  reaction (gc- $i$ ) are a straightforward generalization of the first reaction. As before, the  $\mu$  signal must initiate the first strand



**Fig. 2.** An example of signal molecules (top two left strands) and the transformer, consisting of auxiliary strands (top two right strands) and a saturated template strand (bottom complex) associated with the forward direction of  $0_1 \rightleftharpoons 1_1$ . In this and later figures, the Watson-Crick complement of a domain  $x$  is denoted by  $x^*$ . The state of the system shown is  $0_1$ .

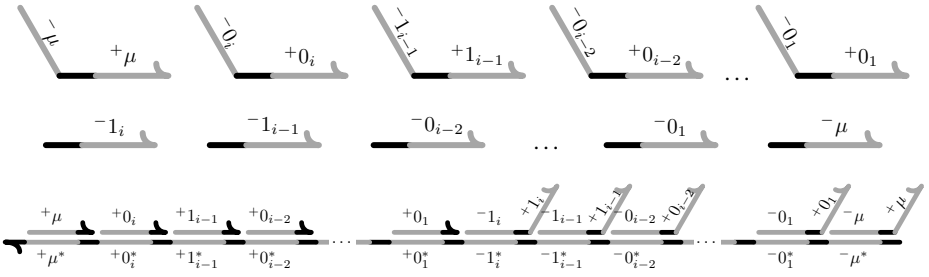


**Fig. 3.** The sequence of strand displacement events for the reaction  $0_1 \rightleftharpoons 1_1$

displacement, and is not produced until the last strand displacement. The number of required intermediate strand displacement reactions is dependent on the number of reactants and products. Specifically, the  $i^{\text{th}}$  reaction requires  $2i + 2$  strand displacements to complete. An example of the transformer for the  $i^{\text{th}}$  reaction is given in Fig. 4.

### 2.3 Correctness

In the reactions of our counter, strand displacement should only happen when the toe-hold of the invading strand first binds to a free toehold, following which a domain of the invading strand displaces the bound domain of the strand being released. The invading and released domains should be identical. We say that such a strand displacement is *legal*. Illegal strand displacements can arise when the invading domain is different from the released domain; we call such displacements *mismatched domain displacements*. Illegal strand displacements can also arise due to blunt-end displacement, *i.e.*, displacements where invading and released domains are identical but domain displacement is not preceded by the binding of a free toehold, or when more than one invading domain strand displaces the strand being released. The next lemma shows correctness of the GRAY counter, assuming that only legal strand displacements can occur.



**Fig. 4.** An example of the signal molecules and the transformer for the  $i^{\text{th}}$  reaction. The counter is in state  $b_n \dots b_{i+1} 0_i 1_{i-1} 0_{i-2} \dots 0_1$ .

**Lemma 2.** *The above strand displacement implementation of the GRAY counter ensures all chemical reactions occur as transactions, and therefore appear atomic, assuming all initial signal molecules exist as single copies and all strand displacements are legal.*

*Proof.* We argue by induction on the sequence of chemical reactions. Prior to any chemical reaction beginning, we require the following invariant to hold: (i) for each digit, there is exactly one available signal which denotes its value, (ii) all template strands of all transformers are saturated, and require the mutex signal  $\mu$  to initiate the first strand displacement, and (iii) there is exactly one available copy of  $\mu$ . The invariant is trivially satisfied for the base case, when no reaction has yet occurred. Suppose the first  $i - 1$  reactions appear atomic, and the invariant is satisfied. Without loss of generality, suppose the next attempted reaction involves the  $k^{\text{th}}$  transformer.

Because we assume that all strand displacements are legal, no auxiliary strand or signal molecule representing the value of a digit can displace any strand in any transformer. Since there is exactly one available copy of the mutex signal,  $\mu$ , that strand alone can initiate a reaction. Suppose the reaction is in the forward direction, as the reverse direction is symmetric. The signal  $\mu$  must initiate the first strand displacement by binding to the left end of the  $k^{\text{th}}$  transformer’s template strand. This begins the transaction. Note that there is another copy of  $\mu$  sequestered at the right end of the template. When the signal  $\mu$  is once again produced, there are two cases to consider.

*Case 1.* If the copy on the right end of the transformer is released, then the transaction succeeded and the counter is in a new state. Furthermore, the invariant is preserved as (i) exactly  $k$  signals that represent  $k$  different digits were consumed and exactly  $k$  signals corresponding to the same  $k$  digits were produced; (ii) the  $k^{\text{th}}$  transformer is saturated, and only a  $\mu$  signal strand can initiate a new reaction on the right end of the template, and (iii) exactly one signal  $\mu$  was produced as the final strand displacement.

*Case 2.* Otherwise, the original copy of  $\mu$  was released, the transaction failed, and the counter is in the same state, satisfying the invariant, as any intermediate strand displacements must have been reversed prior to the original  $\mu$  signal molecule being released.

Importantly, whether or not a transaction succeeds, while one is in progress no other reaction can be initiated since no  $\mu$  signal is available. Thus, all reactions are implemented as transactions and appear atomic.  $\square$

We assume that blunt end displacement and displacement of a single strand by multiple strands do not occur; we do not know how to design strands so as to prevent such illegal displacements. However, we can ensure that the probability of mismatched domain errors is low by ensuring that the energy barrier of a mismatched strand displacement is high. Briefly, the rate at which one domain  $d$  displaces another domain  $d'$  is  $2^{-\Omega(\text{eb}(d,d'))}$ , where  $\text{eb}(d,d')$  is the energy barrier required for  $d$  to displace  $d'$ . More concretely, suppose that all domains have the same length. We consider a simple energy model in which  $\text{eb}(d,d') = |d| - l$ , where  $l$  is the length of the longest subsequence that is common to both  $d$  and  $d'$ . Intuitively, if the bases in the longest common subsequence of  $d$  and  $d'$  form base pairs, then the energy barrier is the total number of base pairs lost when  $d$  displaces  $d'$ .

To ensure that the energy barrier between any pair of distinct domains is sufficiently high, we can use an error correcting code of Schulman and Zukerman [14]. They show how to construct a set of  $2^{\Theta(n)}$  domains (*i.e.*, binary strings in their code) of equal length  $\Theta(n)$  such that the energy barrier between any pair of domains is at least  $cn$ , for any given constant  $c$ . For our  $n$ -bit counter, we use a polynomial number of words in such a code for our signal domains (specifically, as explained in Lemma 3 the GRAY counter uses  $\Theta(n)$  domains, and as explained in Section 2.5, the GRAY-FO counter uses  $\Theta(n^2)$  domains). We choose a sufficiently large constant  $c$ , so that the rate of mismatched domain displacements is at most  $2^{-c'n}$ , where  $c'$  is a constant that depends on  $c$  but is independent of  $n$ .

Since the expected time of the unbiased random walk that simulates our counter is  $\Theta(n^3 2^{2n})$  (see the end of Section 2.4), the walk completes within time  $2^{3n}$  with probability  $1 - 2^{\Theta(n)}$ . By choosing  $c$  so that  $2^{-c'n} < 2^{-4n}$ , we can conclude the probability of mismatched displacements occurring before the unbiased random walk that simulates our counter is completed is an exponentially small function of  $n$ .

## 2.4 Waste and Haste Analysis of the GRAY Counter

Here we analyze the waste—the total number of nucleotide bases of all species consumed and haste—expected time—of the GRAY counter as it advances from initial to final states. We assume single copies of the initial signal, transformer, and mutex species. To analyze waste we first count the number of bases required for all initial signal, transformer, and mutex molecules.

**Lemma 3.** *The total number of nucleotide bases needed for a single copy of each initial signal, transformer, and mutex molecule of the  $n$ -bit GRAY counter is  $\Theta(n^3)$ .*

*Proof.* Each signal species  $0_i$  and the initial mutex signal  $\mu$  is composed of a toehold and two long domains. The same is true of the molecules for states  $1_i$  and the sequestered  $\mu$  signals that are part of the initial transformer species. There is an auxiliary transformer strand species consisting of one toehold and one long domain for each type of signal species. As noted in Section 2.3, to avoid mismatched strand displacement, we choose the  $\Theta(n)$  domains of the signal species according to the Schulman and Zukerman code [14], and so they have length  $\Theta(n)$ . We choose the toehold length to be  $\Theta(1)$ . Since the domain length dominates the toehold length, the total number of bases in all signal species and auxiliary strands is  $\Theta(n^2)$ .

The template molecules in the sets  $\mathcal{T}_i^f$  and  $\mathcal{T}_i^r$  have  $\Theta(i)$  domains, which dominate their length, and thus the template molecules have length  $\Theta(in)$ . Thus, the total number of bases in all transformer molecules in the system is  $\sum_{i=1}^n \Theta(in) = \Theta(n^3)$ .  $\square$

The next lemma shows that just one copy of each signal, mutex, and transformer species is sufficient for the GRAY counter to advance from its initial to final states. The proof is omitted, but is straightforward.

**Lemma 4.** *Advancement of the GRAY counter from its initial to final states requires just one initial copy of the mutex  $\mu$ , one initial copy of each signal species  $0_i$ , and one initial copy of each transformer species  $\mathcal{T}_i^f$ , for each  $i$ ,  $1 \leq i \leq n$ .*

In summary, just one copy of each signal, mutex, and transformer molecule is needed for the counter. Moreover, the total number of bases of these molecules is dominated by those in the transformers and is thus  $\Theta(n^3)$ . Hence the waste is  $\Theta(n^3)$ .

Next consider the expected time (haste) for the counter to progress from its initial to final states. We assume that reactions occur in a volume of size  $\Theta(n^3)$ , since this is the total number of bases of species in the system. Each strand displacement step involves interaction between two species and thus the rate of each strand displacement step is  $1/\Theta(n^3)$ .

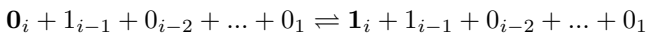
First, consider the *shortest path* from the initial state to the final state. On this path, each order- $i$  reaction is applied  $2^{n-i}$  times and involves  $\Theta(i)$  strand displacements. Thus the total number of strand displacement steps along the shortest path is  $\sum_{i=1}^n \Theta(i)2^{n-i} = \Theta(2^n)$ .

Because each reaction is reversible, the system does not strictly follow the shortest path but rather proceeds as an unbiased random walk along this path. The expected number of steps for a random walk to reach one end of a length- $\Theta(2^n)$  path from the other is  $\Theta((2^n)^2) = \Theta(2^{2n})$  [3]. Therefore, the expected number of strand displacement steps is  $\Theta(2^{2n})$ . Since each strand displacement step occurs at a rate of  $1/\Theta(n^3)$ , the overall expected time—the haste—is  $\Theta(n^3 2^{2n})$ .

## 2.5 A Fixed Order Implementation of the GRAY Counter

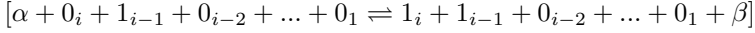
An  $n$ -digit GRAY counter can perform a computation having length exponential in  $n$ , while only generating waste polynomial in  $n$ . However, it relied on template strands containing  $O(n)$  domains, each of length  $O(n)$ , resulting in an overall length of  $O(n^2)$  bases. Synthesis of long nucleic acid strands is challenging, and the fidelity of synthesized strands generally decreases as sequence length increases. For this reason, it is desirable to bound the length of all strands in the system to  $O(n)$  bases. We now briefly describe how a template strand from the GRAY counter consisting of  $2i + 2$  domains, can be split into  $i + 1$  template strands requiring 4 domains each, for any  $i > 1$ . The overall waste will only be increased by a constant, resulting in the same volume, and thus the same haste.

The transformation is straightforward and to simplify the description, we introduce some notation. Consider the (gc- $i$ ) reaction of the GRAY counter which has  $i$  reactants and  $i$  products:

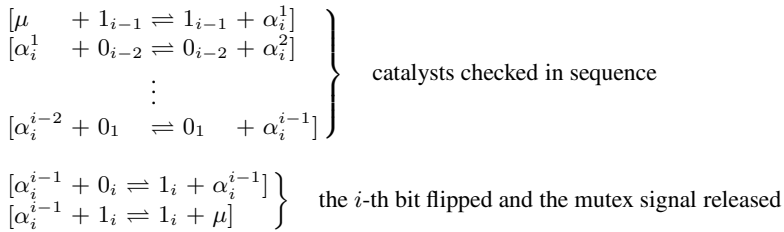




The previous implementation demonstrated that by using the QSW construction and introducing a mutex molecule  $\mu$ —thus creating an order  $i + 1$  reaction—chemical reactions occur as transactions and therefore appear atomic. Specifically,  $\mu$  is first consumed, then  $0_i$ , then,  $1_{i-1}$ , and so on. Likewise, after all reactants are consumed,  $1_i$  is first produced, then  $1_{i-1}$ , and so on, until finally  $\mu$  is produced. We denote a strand displacement implementation supporting a transaction of this type, which is initiated by consuming a mutex  $\alpha$ , and terminated when producing a mutex  $\beta$ , by:



In the case of the GRAY counter,  $\alpha = \beta = \mu$ . Our goal is to convert this order  $i + 1$  reaction into a cascade of  $i + 1$  bi-molecular reactions, while preserving the appearance of atomicity. Using the above notation, we implement the following reaction cascade:



The overall transaction has been split into a cascade of sub-transactions. Each sub-transaction is implemented as a bi-molecular reaction using the QSW construction (see Fig. 3). The first  $i - 1$  sub-transactions check, in sequence, that all  $i - 1$  catalysts are present. The mutex signal  $\mu$  is consumed during the first check. The last two transactions will first perform the bit flip and then releases the mutex signal. Every sub-transaction, except the  $(i - 1)$ -st and the  $i$ -th, produces a unique mutex that is required to initiate the next sub-transaction in the cascade. Upon successful completion of the first  $i$  sub-transactions in the cascade, the final sub-transactions occurs, producing the original mutex signal  $\mu$ , and thus finalizing the overall transaction. The implementation works in the reverse direction in a similar way with the exception that the bit is flipped first and the mutex signal  $\mu$  is released only after the presence of all catalysts have been verified. Using the above transformation for all higher-order reactions in the original GRAY counter implementation results in a new, fixed order counter, GRAY-FO.

## 2.6 The QSW Counter

The stack machine construction of Qian *et al.* [9], which is based on strand displacements systems, can be used to implement a binary counter. We call such a counter a QSW (Qian-Soloviechik-Winfree) counter. An  $n$ -bit implementation advances deterministically through  $2^n$  states and uses reactions of order 2 (some of which involve polymer extension reactions that realize the stack). The transformer molecules used in the strand displacement realizations of these reactions can serve as fuel, biasing the reaction so that the counter advances. We analyze the biased version of the counter; the unbiased version is slower. The expected number of reactions for the biased counter

to advance to its final state is  $\Theta(2^n)$ . Each reaction consumes a constant number of molecules and so the overall expected consumption, or waste, is  $\Theta(2^n)$ . The expected time depends on the volume in which the reaction takes place. If all strands consumed are initially present in the reaction volume, then the volume is  $\Theta(2^n)$  and thus each step takes expected time  $\Theta(2^n)$ , leading to an overall expected time of  $\Theta(2^{2n})$ . Alternatively, it may be possible that fuel concentrations could be replenished over the course of the reaction and waste molecules removed, in which case the volume could be as low as  $\Theta(n)$  and the overall reaction time would be  $\Theta(n2^n)$ . How to replenish fuel or remove waste is not addressed by Qian *et al.*

### 3 Limits on Strand Recycling for Multiple-Copy Systems

In this section, we show that certain classes of chemical reaction systems that efficiently recycle strands, or that can perform useful computations for time that significantly exceeds the number of signals, cannot work properly when multiple copies of the initial species are present. In particular, our GRAY counters do not work in a multi-copy setting.

The underlying problem is the representation of the state of the system as specific *combinations* of signals. If there are multiple copies of the system in the same reaction vessel—as would typically occur in a laboratory setting—then the states of the different copies may interfere with one another. To illustrate this point, we again consider the 3-bit GRAY counter. Initially, in a single copy of the construction, the signals  $\{0_3, 0_2, 0_1\}$  denote the state  $0_30_20_1$ . Consider a two-copy system where the initial set of present signals is duplicated, yielding the multiset  $\{0_3, 0_3, 0_2, 0_2, 0_1, 0_1\}$ . As in the single copy case, assume reaction (1) occurs in the forward direction, followed by reaction (2) in the forward direction. The resulting multiset of signal molecules is  $\{0_3, 0_3, 0_2, 1_2, 0_1, 1_1\}$ . In the single copy case, we intend that reaction (1) in the reverse direction will occur next; however, given the current set of present signals in the two-copy case, reaction (3) in the forward direction could instead occur, resulting in the multiset  $\{0_3, 1_3, 0_2, 1_2, 0_1, 1_1\}$ . At this point, a copy of every signal species is present, and any reaction can occur, in either direction. Furthermore, the single copy case required *at least* seven reactions to produce the final state  $1_30_20_1$ , whereas the two-copy case can reach it in three. Crosstalk between the copies has broken the counter.

In the remainder of this section, we treat this problem formally. We define a *chemical reaction system* to be a tuple  $\mathbf{C} = \langle S, \mathcal{R}, S_0, s_{\text{end}} \rangle$ , where

- $S$  is a set of (signal) molecule species.
- $\mathcal{R}$  is a set of *reaction equations*, where each  $R \in \mathcal{R}$  is an ordered pair of sets of signal molecules. Intuitively, a reaction equation  $R = (I, P)$  consumes the signal molecules in  $I$  as *inputs* and produces the signal molecules in  $P$  as *products*. Our formalism is directional to allow modeling non-reversible reactions; a reversible chemical reaction is modeled as two separate elements of  $\mathcal{R}$ , i.e.,  $(I, P)$  and  $(P, I)$ .
- $S_0$  is a multiset of signal molecules initially present.
- $s_{\text{end}} \in S$  is a signal molecule denoting the end of computation<sup>1</sup>.

<sup>1</sup> A computation may have multiple final states. To model this situation, we can let  $s_{\text{end}}$  be produced in all final reactions, in addition to any other signals that may indicate the result of the computation.

An  $x$ -copy version of  $\mathbf{C}$ , denoted  $\mathbf{C}^{(x)}$ , is obtained by duplicating the initial multiset  $S_0$   $x$  times, i.e.,  $\mathbf{C}^{(x)} = \langle S, \mathcal{R}, S_0^{(x)}, s_{\text{end}} \rangle$  where  $S_0^{(x)}$  is a multiset consisting of  $x$  copies of  $S_0$ .

We formalize computations in  $\mathbf{C}$  in the natural manner: Let  $\rho$  be a sequence of reactions  $R_1, R_2, \dots, R_m$  from  $\mathcal{R}$ , where each  $R_i = (I_i, P_i)$ . We define  $\rho$  to be a *trace* of  $\mathbf{C}$  if  $\rho$  induces a corresponding sequence of multisets  $S_0, S_1, \dots, S_m$ , with  $S_0$  being the multiset of initial signal molecules in  $\mathbf{C}$ , and for all  $1 \leq i \leq m$ , we have both  $I_i \subseteq S_{i-1}$  and  $S_i = S_{i-1} - I_i + P_i$ . (We use “ $-$ ” and “ $+$ ” to denote multiset subtraction and union.)

The next definitions help delineate the class of chemical reaction systems for the main result of this section. For a reaction equation  $R = (I, P)$ , consider the signal molecules in  $I - P$ . We dub these input signals *proper*. (The other signal molecules, in  $I \cap P$ , function as catalysts—they are necessary for the reaction, but not consumed.) We define a set of reactions to be  $k$ -*proper* if  $k$  is the maximum number of proper inputs of all reactions in the set. Note that the GRAY counter system is 1-proper. Let  $|S_0|$  be the number of distinct elements in the multiset  $S_0$ .

**Theorem 1.** *Let  $\mathbf{C} = \langle S, \mathcal{R}, S_0, s_{\text{end}} \rangle$  be a 1-proper chemical reaction system. If there exists a trace that produces  $s_{\text{end}}$  in  $\mathbf{C}$ , then for the  $x$ -copy chemical reaction system  $\mathbf{C}^{(x)}$  with  $x \geq |S| - |S_0|$ , there exists a trace that produces  $s_{\text{end}}$  in at most  $(|S| - |S_0| + 1)(|S| - |S_0|)/2$  steps.*

*Proof.* Let  $\rho = R_1, \dots, R_m$  be a sequence of reactions that produces  $s_{\text{end}}$  in the (single-copy) system  $\mathbf{C}$ , and  $S_0, \dots, S_m$  be the corresponding sequence of multisets of signals. Let  $S' = \bigcup_{0 \leq j \leq m} S_j$  denote the set of all signals that occur in the computation. Obviously,  $S' \subseteq \bar{S}$ . Let  $k = |S'| - |S_0| \leq |S| - |S_0|$  denote the number of molecule species produced by  $\rho$  that were not present initially.

The proof is by construction, constructing a trace of the appropriate length for the multi-copy system from the trace  $\rho$  for the single-copy system. The high-level structure of the proof is as follows: First, we project out from  $\rho$  the  $k$  reactions, in order, that first produce each of the new molecule species. From that sequence, we build a trace of the multi-copy system that is the concatenation of  $k$  phases. Each phase adds one more signal molecule to the multiset of signal molecules present, preserves the presence of all signal molecules previously produced, and “consumes” one copy of the initial signal molecules in  $S_0$ . The  $i$ th phase is at most  $i$  reactions long, so the total length of the trace is bounded by  $\sum_{i=1}^k i = (k+1)k/2 \leq (|S| - |S_0| + 1)(|S| - |S_0|)/2$ .

We now formalize the construction of the  $k$  phases. Let  $s_1, \dots, s_k$  be the sequence of signal molecule species from  $S' - S_0$  in order of their first appearances in  $S_1, \dots, S_m$ , and let  $R_{\text{index}(s_j)}$  be the position in  $\rho$  where  $s_j$  was first produced. In other words,  $R_{\text{index}(s_j)}$  is the reaction that produced  $s_j$  for the first time.

The  $k$  phases are constructed to maintain two invariants:

1. After the  $j$ th phase, the multiset of signal molecules contains at least one copy of each signal molecule in  $\{s_1, \dots, s_j\}$ .
2. The trace constructed so far has not relied on the existence of more than  $j$  copies of the initial signal molecules  $S_0$ .

The invariants are vacuously true initially (before any phases). Assuming they are true after  $j - 1$  phases, we construct the  $j$ th phase as follows: the first reaction in the phase is  $R_{\text{index}(s_j)}$ , the reaction that produced  $s_j$  for the first time. We know this reaction can be applied because all of  $\{s_1, \dots, s_{j-1}\}$  are available, as well as the  $j$ th copy of  $S_0$ . This guarantees that the multiset now contains  $s_j$ , and we have relied on only  $j$  copies of  $S_0$ . However, the reaction may have consumed its inputs. In particular, since the system is 1-proper, the reaction consumed at most 1 input signal molecule. If the reaction consumed 0 molecules, or if the 1 molecule is in  $S_0$ , the invariant is maintained and the phase ends. Otherwise, the reaction consumed some  $s_{j'}$ , where  $j' < j$ . To restore  $s_{j'}$  to the multiset, we append the reaction  $R_{\text{index}(s_{j'})}$  to the phase, which is guaranteed to be applicable by the same reasoning. In turn, this reaction might consume an earlier molecule  $s_{j''}$ , with  $j'' < j'$ , necessitating appending  $R_{\text{index}(s_{j''})}$  to the phase, etc. The sequence  $j, j', j'', \dots$  is strictly decreasing, so it can have length at most  $j$ , which bounds the length of the phase to be at most  $j$  reactions long. At the end of the phase, the invariants are preserved.

Concatenating the  $k$  phases produces a trace for the  $k$ -copy chemical reaction system  $\mathbf{C}^{(k)}$ , which produces all of  $\{s_1, \dots, s_k\}$  within  $(k + 1)k/2$  reactions. Since  $k \leq |S| - |S_0|$ , the result follows.  $\square$

Note that Theorem 1 is much stronger than our intuitive notion of crosstalk short-circuiting a computation. It states that with only a linear number of copies, any signal molecule can be produced in at most a quadratic length computation.

We can formalize the intuitive notion of short-circuiting. A system  $\mathbf{C}$  is *x-copy-tolerant* if, for all  $s \in S$ , the length of the shortest trace to produce  $s$  in  $\mathbf{C}$  and in  $\mathbf{C}^{(x)}$  is the same. A system is *copy-tolerant* if it is *x-copy-tolerant* for all  $x$ .

With that definition, we have the following two corollaries based on the fact that if a 1-proper chemical reaction system is  $|S|$ -copy-tolerant, then  $s_{\text{end}}$  can be computed in  $\mathbf{C}$  in the same number of steps as in  $\mathbf{C}^{(|S|)}$ , which is polynomial in  $|S|$  by Theorem 1.

**Corollary 1.** *For any 1-proper chemical reaction system  $\mathbf{C} = \langle S, \mathcal{R}, S_0, s_{\text{end}} \rangle$  that is  $|S|$ -copy-tolerant, if there is a computation that produces a given signal species  $s_{\text{end}}$  in  $\mathbf{C}$ , then there is a computation that produces  $s_{\text{end}}$  in  $\mathbf{C}$  in  $O(|S|^2)$  steps.*

**Corollary 2.** *Let  $\mathbf{C} = \langle S, \mathcal{R}, S_0, s_{\text{end}} \rangle$  be a 1-proper,  $|S|$ -copy-tolerant chemical reaction system. Then the haste of any computation of  $\mathbf{C}$  is bounded by a polynomial function of the waste.*

## 4 Conclusions

In this paper we have introduced the concept of recycling, or molecule reuse, in strand displacement systems and chemical reaction systems. Our  $n$ -bit GRAY counters effectively use recycling to step through  $2^n$  states while consuming, or wasting, molecules whose total number of bases is  $O(n^3)$ . Our GRAY counter strand displacement constructions also introduce the use of a *mutex* strand to ensure that higher-level chemical reactions are executed atomically. Finally, we show limits to recycling: for example, signals representing the final state of our  $n$ -bit counter can be generated using just  $O(n^2)$  reactions when  $\Theta(n)$  copies of the initial species share the same volume.

One weakness of our counter construction is that the number of distinct domains needed is polynomial in  $n$ , the number of bits of the counter. In contrast, a QSW binary counter that is implemented via the stack machine of Qian *et al.* [9] uses just a constant number of domains independent of  $n$ . Is it possible to construct an  $n$ -bit counter that combines the best of the GRAY and QSW counters, *i.e.*, generates waste that is polynomial in  $n$  and uses  $O(1)$  distinct domains? More generally, can *all* computation be realized by strand displacement systems whose waste and haste are within a (small) polynomial factor of the space and time of the computation? Our negative result raises the question as to whether there are alternative strand displacement realizations of certain chemical reaction system classes that generate little waste, say logarithmic in the computation length, and that also behave correctly in the multi-copy setting. We will investigate these questions in future work.

**Acknowledgements.** Thanks to Bonnie Kirkpatrick and to the anonymous reviewers for their very helpful suggestions.

## References

1. Cardelli, L.: Strand algebras for DNA computing. *Natural Computing* 10(1), 407–428 (2001)
2. Doty, D., Kari, L., Masson, B.: Negative interactions in irreversible self-assembly. In: Sakakibara, Y., Mi, Y. (eds.) *DNA 16 2010*. LNCS, vol. 6518, pp. 37–48. Springer, Heidelberg (2011)
3. Feller, W.: *An Introduction to Probability Theory and Its Applications*, vol. 1. Wiley, Chichester (1971)
4. Hongzhou, G., Chao, J., Xiao, S.-J., Seeman, N.C.: A proximity-based programmable DNA nanoscale assembly line. *Nature* 465, 202–205 (2010)
5. Kharam, A., Jiang, H., Riedel, M., Parhi, K.: Binary counting with chemical reactions. In: *Proceedings of the 2011 Pacific Symposium on Biocomputing*, pp. 302–313. World Scientific Publishing, Singapore (2011)
6. Cardelli, L.: Two-domain DNA strand displacement. In: *Proc. of Developments in Computational Models (DCM 2010)*. *Electronic Proceedings in Theoretical Computer Science*, vol. 26, pp. 47–61 (2010)
7. Lund, K., Manzo, A.T., Dabby, N., Michelotti, N., Johnson-Buck, A., Nangreave, J., Taylor, N., Pei, R., Stojanovic, M.N., Walter, N.G., Winfree, E., Yan, H.: Molecular robots guided by prescriptive landscapes. *Nature* 465, 206–210 (2010)
8. Omabegho, T., Sha, R., Seeman, N.C.: A bipedal DNA brownian motor with coordinated legs. *Science* 324(5923), 67–71 (2009)
9. Qian, L., Soloveichik, D., Winfree, E.: Efficient turing-universal computation with DNA polymers. In: Sakakibara, Y., Mi, Y. (eds.) *DNA 16 2010*. LNCS, vol. 6518, pp. 123–140. Springer, Heidelberg (2011)
10. Qian, L., Winfree, E.: A simple DNA gate motif for synthesizing large-scale circuits. In: *J. R. Soc. Interface* (2011)
11. Reif, J.H., Sahu, S., Yin, P.: Complexity of graph self-assembly in accretive systems and self-destructible systems. In: Carbone, A., Pierce, N.A. (eds.) *DNA 2005*. LNCS, vol. 3892, pp. 257–274. Springer, Heidelberg (2006)
12. Rothmund, P.W.K., Winfree, E.: The program-size complexity of self-assembled squares. In: *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, pp. 459–468 (2000)

13. Savage, C.: A survey of combinatorial Gray codes. *SIAM Review* 39(4), 605–629 (1997)
14. Schulman, L.J., Zuckerman, D.: Asymptotically good codes correcting insertions, deletions, and transpositions. *IEEE Transactions on Information Theory* 45, 2552–2557 (1999)
15. Seelig, G., Soloveichik, D., Zhang, D.Y., Winfree, E.: Enzyme-free nucleic acid logic circuits. *Science* 314(5805), 1585–1588 (2006)
16. Shin, J.-S., Pierce, N.A.: A synthetic DNA walker for molecular transport. *J. Am. Chem. Soc.* 126, 10834–10835 (2004)
17. Soloveichik, D., Seelig, G., Winfree, E.: DNA as a universal substrate for chemical kinetics. *Proc. Nat. Acad. Sci. USA* 107(12), 5393–5398 (2010)
18. Venkataraman, S., Dirks, R.M., Rothmund, P.W.K., Winfree, E., Pierce, N.A.: An autonomous polymerization motor powered by DNA hybridization. *Nature Nanotech* 2(8), 490–494 (2007)
19. Winfree, E.: Algorithmic Self-Assembly of DNA. PhD thesis, Caltech (1998)
20. Zhang, D.Y., Seelig, G.: Dynamic DNA nanotechnology using strand displacement reactions. *Nature Chemistry* 3, 103–113 (2011)

# One-Dimensional Staged Self-assembly

Erik D. Demaine<sup>1</sup>, Sarah Eisenstat<sup>1</sup>,  
Mashhood Ishaque<sup>2</sup>, and Andrew Winslow<sup>2</sup>

<sup>1</sup> MIT Computer Science and Artificial Intelligence Laboratory,  
Cambridge, MA 02139, USA

{edemaine, seisenst}@mit.edu

<sup>2</sup> Department of Computer Science, Tufts University,  
Medford, MA 02155, USA

{mishaque, awinslow}@cs.tufts.edu

**Abstract.** We introduce the problem of staged self-assembly of *one-dimensional* nanostructures, which becomes interesting when the elements are labeled (e.g., representing functional units that must be placed at specific locations). In a restricted model in which each operation has a single terminal assembly, we prove that assembling a given string of labels with the fewest stages is equivalent, up to constant factors, to compressing the string to be uniquely derived from the smallest possible context-free grammar (a well-studied  $O(\log n)$ -approximable problem). Without this restriction, we show that the optimal assembly can be substantially smaller than the optimal context-free grammar, by a factor of  $\Omega(\sqrt{n/\log n})$  even for binary strings of length  $n$ . Fortunately, we can bound this separation in model power by a quadratic function in the number of distinct glues or tiles allowed in the assembly, which is typically small in practice.

**Keywords:** context-free grammar, Wang tile, DNA computing, complexity.

## 1 Introduction

*Self-assembly* is the study of how small particles (typically at the nanoscale, where electrostatic forces overwhelm gravity) interact with each other to conglomerate into larger objects. In theoretical computer science, the standard model is the *tile assembly model* [10] in which the system begins with infinitely many copies of certain square tiles, each with specified glues on the four sides, and tiles translate nondeterministically in the plane until they attach to each other at matching glues. This model effectively enables performing computation, but out of simple geometric parts, and at the cost of physical space resulting from the assembly.

The most studied problem in the tile assembly model is to determine the number of distinct tile types required to assemble a given shape (made out of unit squares). An obvious upper bound is the area of the shape, but in many cases fewer tiles suffice, by building computation into the construction. For example,

an  $n \times n$  square requires  $\Theta(\log n / \log \log n)$  distinct tile types (and glues), by embedding a base- $\log n$  counter, while an  $n \times 1$  rectangle requires  $\Theta(n)$  tile types. The most general result is that any shape, scaled by a sufficiently large factor, can be constructed from  $O(K / \log K)$  tile types (and glues), where  $K$  is the Kolmogorov complexity of the shape [9]. Unfortunately, the scale factor is polynomial in the running time of the Turing machine generating the shape, which is at least the area of the shape. So this result does not directly address the tile complexity of a specific shape, though it suggests that it is difficult to characterize.

An alternate approach is offered by *staged self-assembly* [2] in which the system's tile set can change in a sequence of stages, in particular by an experimenter mixing two systems together. In this model, it is possible to make any shape using a constant number of tile types (and glues); as a result, the main objectives are to minimize both the number of mix operations (work for the experimenter) and the number of stages that must be executed sequentially (makespan or wait time). For example, both an  $n \times n$  square and an  $n \times 1$  rectangle can be assembled using  $O(1)$  tiles and glues and  $O(\log n)$  mixes and stages. Again we lack a general characterization of the number of mixes and stages required for a desired 2D shape.

Our personal communication with bioengineers suggests that the staged assembly model is natural and practical, essentially exposing the experimenter's ability to perform actions as part of a computation/assembly. Furthermore, the results are more practical, as it is difficult in practice to design many different glues that attract only in pairs, without any attraction between unpaired glues. Assembling a  $1000 \times 1$  rectangle would be impractical without staging (requiring 1000 tile types and 999 distinct glues), but is extremely practical with staging (requiring only 6 tile types, 3 distinct glues and 10 stages).

In this paper, we aim to characterize the resources required to staged-assemble a *one-dimensional* object. Just making a  $1 \times n$  rectangular shape is trivial, so this direction has so far been overlooked. But in practice, experimenters often want to build an object that not only takes on a desired shape but also carries out a desired functionality. A typical example is to arrange nanodots or bioagents in a particular pattern within a shape.<sup>1</sup> We model this problem as constructing a *labeled* shape, where each unit square has a label within a fixed alphabet, and each tile type used to build the shape also has a label, which must match in construction. Thus the input to the problem is a string of labels, and the goal is to find a staged assembly with few glues, mixes, and stages. In fact we show that four glues and  $O(\log n)$  stages always suffice, so a single objective remains: minimize the mixes.

The problem of computing a minimal tile assembly system that produces a labeled shape has been studied previously. Heuristic approaches have been developed to find the smallest tile set that uniquely assembles an input labeled shape [6,4], and the problem of finding a minimum-size tile set has been shown to be NP-hard [6].

---

<sup>1</sup> Personal communication with Hyunmin Yi, 2008–2010.



We successfully characterize the number of mixes required to staged-assemble a string in two natural situations. In the first setting (Section 4), we restrict mixing operations to produce a single terminal assembly for use in the next mix. This restriction seems to be common to all previous staged-assembly algorithms [2], but we do not know it to be practically motivated. In the second setting (Section 5.5), we allow multiple “parallel assemblies” resulting from a mix (a new though natural idea), and consider the more natural restriction that the number of glues is constant. In either setting, we show that the minimum number of mixes is within a constant factor of the smallest context-free grammar that generates exactly the given string. The latter problem is well-studied, has a polynomial-time  $O(\log n)$ -approximation algorithm based on Lempel-Ziv compression, and likely has no  $o(\log \log n)$ -approximation [1,5,7,8].

We show that our relations are nearly tight by constructing a family of strings (Section 5) showing a separation in power between (unrestricted) staged assembly and context-free grammars. Specifically, an  $n$ -bit binary string can be assembled using  $O(k)$  glues with  $O(k)$  mixes but requires a context-free grammar of size  $\Omega(k^2)$ , for a ratio of  $\Omega(k)$ . Our upper bound shows that the worst-case separation is  $O(k^2)$ . As a function of  $n$  (with an unbounded number of glues), we prove that the ratio is  $\Omega(\sqrt{n/\log n})$ . In practice, small feature sizes make the number of glues typically small, in which case context-free grammars are actually a good approximation to optimal staged self-assemblies.

The labeled 1D staged self-assembly model offers a balance of tractability, being easier than general staged assembly by reducing the dimension to 1, yet harder (and more practical) by adding labels to the target shape. The connections we show to context-free-grammar compression illustrate that the problem is difficult, yet for the case of many glues, still not fully understood. The approximation algorithm resulting from our study is simple and efficient, having been implemented in an online web system,<sup>2</sup> which is currently being considered for practical use by the Tufts Department of Bioengineering, in a setting where labeled 1D assemblies are of significant interest.

## 2 Context-Free Grammars

**Definition 1.** A *context-free grammar (CFG)* is defined as a 4-tuple  $(\Sigma, \Gamma, S, \Delta)$  where  $\Sigma$  is a set of terminal symbols,  $\Gamma$  is a set of non-terminal symbols,  $S$  is a special element of  $\Gamma$  called the start symbol and  $\Delta$  is a set of productions.

Each production consists of a *left-hand side* containing a single non-terminal symbol, and a *right-hand side* containing a (non-empty) sequence of terminal and non-terminal symbols. A CFG *derives* a string by repeated replacement of non-terminal symbols with strings of terminal and non-terminal symbols according to the productions in  $\Delta$ , beginning with the single symbol  $S$ . The *language* of a CFG is the set of derivable strings consisting solely of terminal symbols.

---

<sup>2</sup> <http://selfdisassembler.appspot.com/>

**Definition 2.** The size of a context-free grammar  $G$ , denoted  $|G|$ , is the total number of symbols appearing on the right-hand sides of the productions in  $G$ .

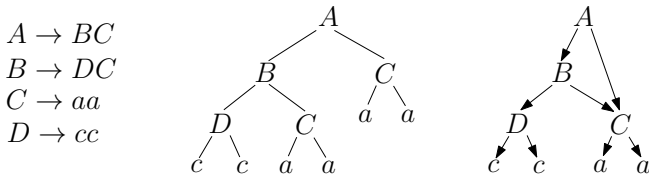
Note that this definition counts the *total* number of symbols, so symbols appearing multiple times contribute to the count multiple times. In this paper we consider only CFGs that are *deterministic* (with only one production per left-hand side) and generate exactly one string.

**Definition 3.** A restricted context-free grammar (written RCFG) is a CFG which is deterministic and has a language consisting of a single string.

For an RCFG  $G$  deriving a string  $s$ , the *parse tree* of  $G$  is the tree created by beginning with a single node with label  $S$  (the start symbol), and adding children to a leaf node for each production applied. The result is a tree where each internal node is a non-terminal symbol, each leaf node is a terminal symbol, and an in-order traversal of the leaves gives the string  $s$ .

**Definition 4.** The smallest grammar problem is the following: given an input string  $s$ , find the smallest RCFG deriving  $s$ .

Any RCFG  $G$  has exactly one parse tree. Each internal node in the parse tree has a corresponding non-terminal symbol from  $G$ . If we merge all such nodes with the same non-terminal, the result is a *directed acyclic graph* (DAG) called the *parse DAG*. See Figure 1 for an example of an RCFG and its parse tree and parse DAG.



**Fig. 1.** A restricted context-free grammar (RCFG) and its corresponding parse tree and parse DAG

### 3 Staged Self-assembly

In this section we describe the 1D labeled staged self-assembly system model. The model described here is a variant of the staged self-assembly model defined in [2]. In this model, individual building blocks are 2D square-shaped *tiles* that translate in the plane. Each tile has four sides (north, south, east, and west) and has *glues* on its east and west sides. Each tile also has a label ( $a, b, c, \dots$ ). We denote a tile  $x_1[x_2]x_3$  where  $x_1$  is the west glue,  $x_2$  is the label,  $x_3$  is the east glue (e.g.,  $1[a]2$ ).

Tiles combine when the pair of glues on the west side of one tile and east side of the other are *complementary*. We denote glue values by numbers (e.g.,  $1, 2, \dots$ ),

and the complementary glues are denoted  $1', 2', \dots$ . In this paper we use the convention that east glues are always complementary (prime) glues. So a tile  $1[a]2$  actually has east glue  $2'$ .

When tiles combine they create *assemblies* (and we consider tiles to be a special case of assemblies). The labels of each individual tile combine to form a *label string* of the assembly consisting of the labels of the combined tiles in order. For example,  $1[a]2$  and  $2[b]3$  combine to form the assembly  $1[ab]3$ . Note that the east glue of  $1[a]2$  and west glue of  $2[b]3$  have disappeared: they are on the interior of the assembly and are omitted for clarity. Assemblies can also be combined to form larger assemblies. The size of an assembly is the number of tiles it contains.

Initially each tile type exists in a separate *bin*. When bins are mixed, the assemblies present in each bin are free to attach to each other. The products of each mixing are *terminal assemblies*: assemblies that do not attach to any other assemblies. All other assemblies produced during the mixing are assumed to be filtered out before the bin is combined with other bins.

A self-assembly system instance is defined by the starting tiles and a mix DAG defining bins and the orders in which they are mixed. The mix DAG is a *rooted DAG*: a DAG with only one node (the *root*) without in-edges, where each node represents a bin and the edges leaving it point to the bins whose contents are mixed into this bin. Each leaf of the DAG is a bin of a single tile type.

**Definition 5.** *A self-assembly system (SAS) is a one-dimensional labeled staged self-assembly instance that produces a single goal assembly and is defined by a mix DAG and a unique tile type for each leaf of the DAG.*

**Definition 6.** *The size of a SAS  $A$  (denoted  $|A|$ ) is the number of edges in its mix DAG.*

The goal assembly produced by a SAS must appear in the bin corresponding to the root node of the mix DAG. Reading the labels of an assembly from west to east defines a string which we call the *label string* of the assembly. The label string of the goal assembly is the string *generated* by the SAS.

In previous staged-assembly constructions [2], each bin has a single assembly produced in it by mixing the contents of two other bins (which also contain single items). However the model as defined does not require that each bin contains a single assembly. A mix DAG in which one or more bins has multiple assemblies is said to use *bin parallelism*. We distinguish a self-assembly system instance that does not use bin parallelism as a *single self-assembly system* (SSAS).

**Definition 7.** *A single self-assembly system (SSAS) is a SAS in which no bin contains more than one distinct assembly.*

**Definition 8.** *The minimum SSAS problem is the following: given an input string  $s$ , find a smallest SSAS generating an assembly with label string  $s$ .*

## 4 Equivalence between RCFGs and SSASs

In this section we show that converting between an RCFG  $G$  deriving a string  $s$  and a SSAS instance  $A$  assembling a labeled assembly with label  $s$  is possible with only a constant-factor scaling. As a result, any algorithm generating an  $O(f(n))$ -approximation to *either* the minimum grammar problem or the minimum SSAS problem implies an  $O(f(n))$ -approximation algorithm for the other.

### 4.1 Converting RCFGs to SSASs

Let  $G$  be an RCFG deriving a string  $s$ . We begin by converting  $G$  to an equivalent RCFG  $G'$  with at most two symbols on the right-hand side of each production (such a CFG is called a *binary CFG*).

Recall that each rule is represented by a subtree of the parse DAG consisting of a root node (the left-hand side symbol) and its children (the right-hand side symbols). This subtree can obviously be converted into a binary tree which has size at most twice the number of leaves, and is at most twice the size of the original subtree. So each rule can be expanded to a set of binary rules with at most twice as many symbols. As a result,  $G$  is at most doubled in size and thus  $|G'| \leq 2|G|$ .

Next we convert each production of  $G'$  to a SSAS mixing. However, a problem occurs if the same non-terminal appears as a right-hand side symbol in several production rules. Recall that a production in the grammar specifies the left-to-right order in which the right-hand side symbols appear, while the west-to-east order in which assemblies attach is determined by their glues. To produce exactly the assembly desired in a mixing requires combining its subassemblies *with the correct glues*.

To resolve this issue, we construct several copies of every assembly: one for each possible west/east glue pair. Since the grammar is binary, at most two assemblies are mixed in each bin and so three glue pairs is enough to uniquely specify the mixing product. Given a production  $A \rightarrow BC$ , we create six bins and six mixings that assemble the six west/east glue pair combinations for  $A$  from the six west/east glue pair combinations for  $B$  and  $C$  (see Table 1).

**Lemma 1.** *A parse DAG for a binary RCFG  $G'$  deriving string  $s$  can be converted to a valid SSAS  $A$  of size at most  $6|G'|$  that constructs an assembly with label string  $s$ .*

**Table 1.** The set of mixings to produce all necessary glue pair variations for assembly  $A$  in the production  $A \rightarrow BC$

A glues	(1, 2)	(1, 3)	(2, 1)	(2, 3)	(3, 1)	(3, 2)
B glues	(1, 3)	(1, 2)	(2, 3)	(2, 1)	(3, 2)	(3, 1)
C glues	(3, 2)	(2, 3)	(3, 1)	(1, 3)	(2, 1)	(1, 2)

*Proof.* We build the mix DAG of  $A$  in the following way: For each symbol (terminal and non-terminal) create 6 bins for the glue-pair variants of the symbol. For each production  $A \rightarrow BC$  of  $G'$ , mix the 6 bins of  $B$  and  $C$  into the 6 bins of  $A$  as in Table 1. The resulting mix DAG has 6 bins for each symbol of  $G'$ , each containing a unique glue-pair variant of an assembly with label string corresponding to the string derived by the symbol in  $G'$ . Each edge of the parse DAG of  $G'$  is converted to 6 edges in the mix DAG of  $A$ , one for each glue-pair variant. So  $|A| \leq 6|G'|$ .  $\square$

**Theorem 1.** *Given an RCFG  $G$  deriving a string  $s$ , the algorithm described in Section 4.1 computes a SSAS instance  $A$  with  $|A| \leq 12|G|$  that produces an assembly with label string  $s$ .*

*Proof.* The algorithm converts  $G$  to a binary RCFG  $G'$ , and then converts  $G'$  to a mix DAG for  $A$ . By Lemma 1,  $|A| \leq 6|G'|$ . So  $|A| \leq 6|G'| \leq 12|G|$ .  $\square$

## 4.2 Converting SSASs to RCFGs

Let  $A$  be a SSAS constructing an assembly with label string  $s$ . We perform a (nearly) one-to-one mapping from the mix DAG of  $A$  to the parse DAG of a grammar  $G$ . For each leaf bin of  $A$ , create a terminal symbol in  $G$  equal to the label string of the tile in the bin. For each non-leaf bin of  $A$ , create a non-terminal symbol in  $G$ . For each mixing in  $A$  combining the contents of bins  $b_1, b_2, \dots, b_k$  into bin  $B$ , create a production in  $G$  with  $B$  on the left-hand side and  $b_1$  through  $b_k$  on the right-hand side in the order they combine when mixed in  $B$ .

**Theorem 2.** *For any SSAS  $A$  constructing an assembly with label string  $s$ , an RCFG  $G$  deriving  $s$  can be constructed from  $A$  such that  $|G| = |A|$ .*

*Proof.* The terminal symbols of  $G$  are equal to the label strings of their corresponding tiles. Each mixing in  $A$  produces a single assembly with a label string equal to the string derived by the corresponding non-terminal symbol in  $G$ , because the production orders the right-hand side symbols in the same order that they combine in  $A$ . So the start symbol of  $G$  derives a string equal to the label string of the assembly produces in the root of the mix DAG of  $A$ . So  $G$  derives  $s$ . Each edge of the mix DAG of  $A$  causes a right-hand side symbol to appear in a production of  $G$ . So  $|G| = |A|$ .  $\square$

## 4.3 Approximation Equivalence

The conversions presented above in Sections 4.1 and 4.2 immediately imply that approximation algorithms for either problem transfer to the other, at a constant-factor loss.

**Corollary 1.** *An  $O(f(n))$ -approximation algorithm for the smallest grammar problem exists if and only if an  $O(f(n))$ -approximation algorithm for the minimum SSAS problem exists.*

In practice this theorem makes computing efficient SSAS instances easier, as several  $O(\log n)$ -approximation algorithms to the minimum grammar problem exist [1,7,8]. We have taken advantage of this fact to produce a software tool for finding  $O(\log n)$  approximations to the minimum SSAS problem in  $O(n)$  time using the algorithm by Sakamoto [8].

This result also suggests that finding an improved approximation algorithm for the minimum SSAS problem is unlikely. In 2002, Lehman showed that a polynomial-time approximation algorithm for the smallest grammar problem with factor  $o(\frac{\log n}{\log \log n})$  would enable progress on “a difficult algebraic problem in a well-studied area” [5].

## 5 Separation Between SASs and RCFGs

Now we show that the general 1D staged self-assembly model (SAS) is *not* equivalent to RCFGs. The proof is constructive: we give a set of strings and describe a set of SAS instances that produce assemblies with these label strings. We then show that any CFG producing these strings is asymptotically larger than the SAS instance producing the corresponding label string.

It might appear obvious that allowing bin parallelism should allow a reduction in the amount of work needed to construct an assembly. However, using parallelism has two costs that make saving work difficult. First, for any minimal SAS, no two assemblies in the same bin may share a common glue (this is proven in Lemma 3). As a result, additional parallelism requires more unique glues, which in turn requires more starting bins, and thus more work. Second, since the goal of an assembly system is to construct a single goal assembly, bins with parallelism must eventually be “collapsed” into a single bin with a single object (otherwise the parallelism was extraneous). Collapsing bins with parallelism involves adding tiles to join the various assemblies together, and since the glues on each assembly are unique, creating and mixing the joining tiles requires additional work proportional to the amount of parallelism in the bin.

### 5.1 A Set of Strings $S_k$

To derive an asymptotic bound between SASs and RCFGs, we use a special set of strings that can be built by small SASs but require large RCFGs. Each string consists of a sequence of *interleavings* of pairs of smaller strings. We will consider only odd values of  $k$  for the remainder of the paper.

Let  $\text{BINARY}(i, \ell)$  be the binary representation of  $i$  of length  $\ell$ . The following is a function used to double every character in a string:

$$\text{DOUBLE}(b_1 b_2 b_3 \dots b_n) = b_1 b_1 b_2 b_2 b_3 b_3 \dots b_n b_n$$

We define  $s_1 \circ s_2$  to be the concatenation of string  $s_1$  followed by  $s_2$ . We wish to encode a number of distinct “characters” in binary. To construct a suitably hard-to-compress string, we want to ensure that the beginning and end of each encoding are clearly delineated. To that end, we define the following strings for all values of  $k$  and all values of  $i < 2k$ :

**Definition 9.**  $A_{k,i} = (01) \circ \text{DOUBLE}(\text{BINARY}(i, 1 + \lceil \log k \rceil)) \circ (01)$ .

Note that each such string has length  $6 + 2\lceil \log k \rceil$ .

We wish to use these characters to construct a string with a lot of structure (so that it is efficiently constructible using a SAS) but minimal repetition (so that it is not efficiently constructible using a CFG). To minimize repetition, we choose a string with the property that no sequential pair of characters is repeated. We define the following functions, which are permutations for  $0 \leq x < k$ :

$$\pi_{k,0}(x) = 2x \bmod k \qquad \pi_{k,1}(x) = 2x + 1 \bmod k$$

We use these two simple functions to construct a more complex permutation.

**Definition 10.** Say that the bits of  $\text{BINARY}(i, \ell)$  are  $b_1, \dots, b_\ell$ . Then

$$\Pi_{k,\ell,i}(j) = \pi_{k,b_\ell}(\pi_{k,b_{\ell-1}}(\dots \pi_{k,b_2}(\pi_{k,b_1}(j)) \dots)).$$

Because  $k$  is odd, this function is a permutation for  $0 \leq j < k$ . In addition, as long as  $0 \leq i < 2^\ell$ , this function has the property that  $\Pi_{k,\ell,i}(j) = (2^\ell \cdot j + i) \bmod k$ . This means that for fixed values of  $k, \ell$ , and  $j$ , each value of  $i$  such that  $0 \leq i < k$  will generate a different value of  $\Pi_{k,\ell,i}(j)$ .

This permutation can be used to ensure that no sequential pair of characters is repeated. To do so, we construct pairs of characters as follows:

$$C_{k,i,j} = A_{k,j} \circ (01)^{\lceil \log k \rceil} \circ A_{k,k+\Pi_{k,\lceil \log k \rceil,i}(j)} \circ (01)^{\lceil \log k \rceil}.$$

We concatenate these pairs to construct  $P_{k,i} = C_{k,i,0} \circ C_{k,i,1} \circ \dots \circ C_{k,i,k-1}$ . Note that the length of each  $C_{k,i,j}$  is  $12 + 8\lceil \log k \rceil$ , and therefore the length of each  $P_{k,i}$  is  $(12 + 8\lceil \log k \rceil) \cdot k$ .

We concatenate each  $P_{k,i}$  to get the string we wish to compress:

**Definition 11.**  $S_k = 01 \circ P_{k,0} \circ 01 \circ P_{k,1} \circ 01 \circ \dots \circ 01 \circ P_{k,k-1} \circ 01$

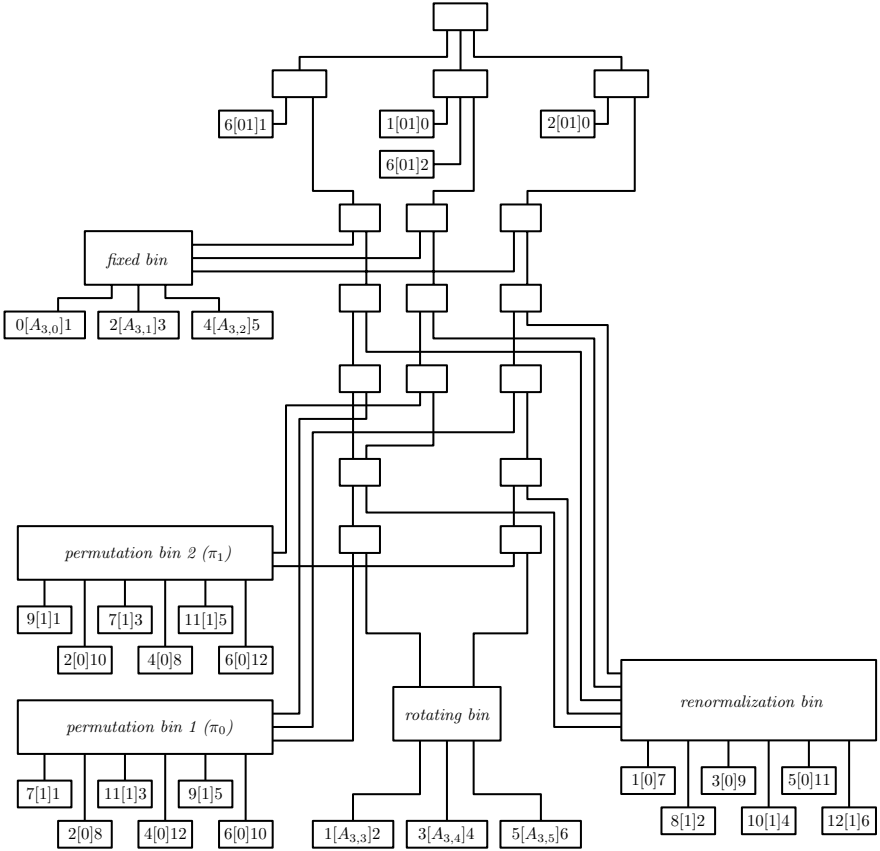
In the next two subsections we give bounds on compressing  $S_k$  using both a RCFG and a SAS.

### 5.2 A SAS Upper Bound for $S_k$

Now we describe a self-assembly system using bin parallelism that produces an assembly with  $S_k$  as its label string. The system is broken down into several subsystems described in this section. A diagram of the SAS for  $S_3$  is seen in Figure 2.

**Constructing  $A_{k,i}$  for all  $0 \leq i < 2k$ .** Say that we are given  $2k$  glue pairs  $x_i, y_i$ , and that we want to assemble  $x_i[A_{k,i}]y_i$  for each  $0 \leq i < 2k$ . In addition, say that we are given three additional glues  $g_0, g_1$ , and  $g_2$  for use in our construction. Let  $\ell = 1 + \lceil \log k \rceil$ .

For each binary string  $s$  of length  $\leq \ell$ , we construct two bins:  $I_s$  and  $F_s$ . Let  $s = t \circ b$ , where  $b \in \{0, 1\}$ .  $I_s$  will contain an assembly with glue  $g_0$  on the left,



**Fig. 2.** The mix DAG for a SAS generating an assembly with label string  $S_3$

glue  $g_1$  on the right, and the label  $\text{DOUBLE}(t) \circ b$ .  $F_s$  will contain an assembly with glue  $g_0$  on the left, glue  $g_2$  on the right, and the label  $\text{DOUBLE}(s)$ . The assembly in bin  $I_s$  will be constructed by adding the tile  $g_2[b]g_1$  to the assembly in bin  $F_t$ . The assembly in bin  $F_s$  will be constructed by adding the tile  $g_1[b]g_2$  to the assembly in bin  $I_s$ .

To finish this construction, we add the constant-sized assemblies  $x_i[0]1g_0$  and  $g_2[0]1y_i$  to the bin  $F_{\text{BINARY}(i,\ell)}$ . This ensures that for  $0 \leq i < 2k$ , the bin  $F_{\text{BINARY}(i,\ell)}$  contains an assembly with the label  $A_{k,i}$ . The total number of bins required for this construction is  $\Theta(k)$ .

**Fixed and Rotating Bins.** The fixed bin contains the following set of tiles:

$$0[A_{k,0}]1, 2[A_{k,1}]3, \dots, (2k-2)[A_{k,k-1}](2k-1)$$

The rotating bin contains the following set of tiles:

$$1[A_{k,k+0}]2, 3[A_{k,k+1}]4, \dots, (2k-1)[A_{k,k+(k-1)}](2k)$$



**Permutation and Renormalization Bins.** Permuting the assemblies in the rotating bin is simulated by attaching *permutation tiles* to the east and west ends of those assemblies. The permutations  $\pi_{k,0}$  and  $\pi_{k,1}$  are implemented as two sets of  $2k$  tiles, each set in a separate *permutation bin*. A third set of  $2k$  tiles are put in a *renormalization bin* used to solve a technical issue with the permutation bins.

The permutation bin for  $\pi_{k,0}$  has tiles that replace the primal glues of assembly  $i$  ( $2i+1$  and  $2i+2$ ) with the dual glues of assembly  $\pi_{k,0}(i)$  ( $2\pi_{k,0}(i)+1+(2k+1)$  and  $2\pi_{k,0}(i)+2+(2k+1)$ ) for all assemblies  $0 \leq i \leq k-1$ . The permutation bin for  $\pi_{k,1}$  is constructed analogously. Each tile attaches to either the east or west end of the assembly and correspondingly has the primal and dual glues on its east and west sides. The tiles attaching to the east end of the assembly have the label 0; the tiles attaching to the west end of the assembly have the label 1.

The renormalization bin has a pair of tiles for changing the dual glues of assembly  $i$  ( $(2i+1)+(2k+1)$  and  $(2i+2)+(2k+1)$ ) to its primal glues ( $(2i+1)$  and  $(2i+2)$ ). The tiles attaching to the east end of each assembly have the label 1; the tiles attaching to the west end of each assembly have the label 0.

**Creating Interleaved Assemblies.** The permutation and renormalization bins are applied in a branching manner to produce all permutation sequences of length  $\ell$ . First  $\pi_{k,0}$  and  $\pi_{k,1}$  are mixed separately with the rotating bin, then  $\pi_{k,0}$  and  $\pi_{k,1}$  are each mixed separately with the product of both of these mixings, etc. After each mixing with a permutation bin, the renormalization bin is mixed with the product. After all permutation sequences are created, the fixed bin is mixed with each, creating single assemblies with label strings  $P_{k,i}$  for all  $0 \leq i < k$ .

**Combining Interleaved Assemblies.** The final step is to combine each assembly with label  $P_{k,i}$  into a single assembly. Each assembly is contained in a separate bin after its production, and has glue 0 on its west side and glue  $2k$  on its east side. To the assembly with label  $P_{k,i}$ , the tiles  $(2k+1+i)[1]0$  and  $(2k)[0](2k+2+i)$  are added. Then these assemblies are combined to produce a single long assembly with glue  $(2k+1)$  on the west side, and glue  $(3k+1)$  on the east. To finish off the assembly, two more tiles are added:  $(null)[0](2k+1)$  and  $(3k+1)[1](null)$ . This ensures that the final result is an assembly with the label  $S_k$  and null glues on both ends.

**Theorem 3.** *The SAS described in Section 5.2 has size  $O(k)$ .*

*Proof.* Break the SAS into the following sections:

1. Creating the  $a$ -bin and  $b$ -bin.
2. Creating the permutation and renormalization bins.
3. Creating the interleaved assemblies.
4. Combining interleave assemblies.

Item 1 requires  $O(k)$  edges to create a tile for each symbol in  $A_k$  or  $B_k$  respectively and mixing them together. Item 2 requires  $O(k)$  edges to create three bins

each with a pair of tiles for each  $c$ -buffered element of the  $b$ -bin. Item 3 requires  $O(k)$  edges: this portion of the mix DAG resembles an upside-down tree and contains no more than two leaves per permutation assembly. Item 4 requires  $O(1)$  edges per assembly (and thus  $O(k)$  edges total) to add two location-specifying tiles and combine it with the other assemblies into a single bin. In total  $k$  interleave assemblies (one per shift) are created, so  $O(k)$  edges are in this portion of the mix DAG. Combining interleave assemblies is done by adding at most two tiles to each interleave assembly followed by combining them into a single bin. A constant number of edges exist for each assembly, so  $O(k)$  edges exist in this portion of the mix DAG.  $\square$

### 5.3 An RCFG Lower Bound for $S_k$

The following definition and theorem are taken from [7].

**Definition 12.** *As defined in [3], the size of the LZ-factorization of a string  $s$  (denoted  $|LZ(s)|$ ) is the number of elements generated by the LZ77 algorithm without self-referencing.*

**Theorem 4.** *For an RCFG  $G$  generating a string  $s$ ,  $|LZ(s)| \leq |G|$ .*

**Lemma 2.** *All factors in the LZ-factorization of  $S_k$  have size  $< 16\lceil \log k \rceil + 26$ .*

*Proof.* Assume, for the sake of contradiction, the LZ-factorization of  $S_k$  contains some factor  $y$  of size  $\geq 16\lceil \log k \rceil + 26$ . Then the factor is long enough that there must be some  $i, j$  such that  $C_{k,i,j}$  is a substring of  $y$ . Let  $x$  be the part of the string preceding  $y$ . Then by the definition of LZ factorization,  $y$  is a substring of  $x$ , and therefore  $C_{k,i,j}$  is a substring of  $x$ .

$C_{k,i,j}$  contains as a substring the string  $A_{k,j}$ . To ensure the correct parity on runs of characters, the portion of  $x$  where  $A_{k,j}$  is found must have been completely generated by some other  $A_{k,j^*}$ . Then it must be that  $A_{k,j} = A_{k,j^*}$ , and by Definition 9, it follows that  $j = j^*$ . So the portion of  $x$  where  $C_{k,i,j}$  is found must have been completely generated by some other  $C_{k,i^*,j}$ , where  $i \neq i^*$ . Then  $A_{k,k+\Pi_{k,\lceil \log k \rceil},i}(j) = A_{k,k+\Pi_{k,\lceil \log k \rceil},i^*}(j)$ . By Definition 9, it follows that  $k + \Pi_{k,\lceil \log k \rceil},i(j) = k + \Pi_{k,\lceil \log k \rceil},i^*(j)$ . Therefore, by Definition 10,  $i = i^*$ , which gives us a contradiction.  $\square$

**Theorem 5.** *The smallest CFG that can be used to construct  $S_k$  has size  $\Omega(k^2)$ .*

*Proof.* By Lemma 2, the maximum length of an LZ factor is  $16\lceil \log k \rceil + 29$ . The sum of the lengths of the LZ factors is equal to  $|S_k| = \Theta(k^2 \log k)$ . Hence, the number of LZ factors is  $\Omega(k^2)$ . By Theorem 4, the size of the minimum grammar must therefore be  $\Omega(k^2)$ .  $\square$

### 5.4 Asymptotic Separation of SASs and RCFGs for $S_k$

*Separation* refers to the minimum difference in size between an RCFG and a SAS generating the same (label) string. Here we show the separation achieved for the strings  $S_k$ , where  $k$  is the number of glues used to generate the label string  $S_k$  by the SAS in Section 5.3 and  $n$  is the length of  $S_k$ .

**Corollary 2.** *The strings  $S_k$  have separation  $\Omega(k)$ .*

*Proof.* By Theorem 5, any RCFG generating  $S_k$  has size  $\Omega(k^2)$ . By Theorem 3, a self-assembly system of size  $O(k)$  exists that produces an assembly with label string  $S_k$ . So the ratio of the size of any grammar generating  $S_k$  to the size of some SAS instance is  $\Omega(k)$ .  $\square$

**Corollary 3.** *The strings  $S_k$  have separation  $\Omega(\sqrt{n/\log n})$ .*

*Proof.* The length of  $S_k$  is  $\Theta(k^2 \log k)$ . So  $k = \Theta(\sqrt{n/\log n})$ . By Corollary 2, the separation is  $\Omega(k)$ . So the separation is also  $\Omega(\sqrt{n/\log n})$ .  $\square$

Given that the number of glues is limited in practice, it is natural to consider whether  $\Omega(k)$  separation is possible for  $k$  glues where  $k \ll n$ . We show this is possible for  $k = \Theta(\log n)$ .

**Definition 13.** *Define the recursive string  $T_{k,t} = 01 \circ T_{k,t-1} \circ 01 \circ T_{k,t-1} \circ 01$ , where  $T_{k,1} = S_k$ . The length of  $T_{k,t}$  is  $\Theta(2^t |S_k|) = \Theta(2^t k^2 \log k)$ .*

**Theorem 6.** *The strings  $T_{k,k}$  have separation  $\Omega(k)$  and use  $\Theta(\log n)$  glues.*

*Proof.* Since  $T_{k,k}$  has  $S_k$  as a substring, any CFG generating  $T_{k,k}$  has size  $\Omega(k^2)$  by Theorem 5. To construct a SAS to generate this string, we first use the SAS described in Section 5.2 to generate an assembly  $a[S_k]b$ . We can then add a constant number of tiles to get two assemblies  $c[1S_k 0]e$  and  $e[1S_k 0]d$ , which when combined create the assembly  $c[1S_k 0 1 S_k 0]d$ . We then add two more tiles to construct the assembly  $a[0 1 S_k 0 1 S_k 0 1]b$ . This process can then be repeated  $k$  times. In total  $O(k)$  additional work is performed, so the new SAS has size  $O(k)$ . The length  $n$  of the string is  $\Theta(2^k k^2 \log k)$ , so  $k = \Theta(\log n)$ .  $\square$

## 5.5 Upper Bounds for Separation of SASs and RCFGs

The PSASs described in Section 5.2 constructing  $S_k$  used  $O(k)$  distinct glue pairs to achieve a separation of  $\Omega(k)$ . We now show bounds on the worst-case separation in terms of the number of glues  $k$  and the length of the string  $n$ .

**Lemma 3.** *Given a minimal SAS  $A$ , any two distinct assemblies  $A_1$  and  $A_2$  in the same bin must have different glues on either the west side or the east side.*

*Proof.* For the sake of contradiction, say that there is a distinct pair of assemblies  $A_1$  and  $A_2$  with matching glues on both the west and east sides of the assemblies. Because the accessible glues on both assemblies are identical, any assembly which adheres to  $A_1$  must also adhere to  $A_2$ , and vice versa. Hence, for every superassembly of  $A_1$ , there is a corresponding superassembly of  $A_2$  in the same bin with the same accessible glues, but a different label sequence. Any attempt to merge two such assemblies to create a single assembly results in an infinite label sequence. So the SAS  $A$  cannot produce a single goal assembly, violating the definition of a SAS.  $\square$

**Corollary 4.** *Given a minimal SAS  $A$  using  $k$  glues to produce a string  $s$ , each bin in  $A$  contains at most  $k^2$  distinct assemblies.*

**Lemma 4.** *Given a SAS  $A$  using  $k$  glues and generating an assembly with label string  $s$ , an RCFG of size  $O(k^2|A|)$  generating  $s$  can be constructed.*

*Proof.* For each bin in  $A$  and each distinct assembly in that bin, construct one bin in the SSAS  $B$ . By Corollary 4, the number of bins in  $B$  will be at most  $k^2$  times the number of bins in  $A$ .

Now consider what happens when  $\ell$  bins in  $A$  are simultaneously mixed to produce a single bin  $c$  containing several assemblies. How many edges must we add to  $B$  to ensure that each assembly in  $c$  is correctly constructed in  $B$ ? To determine this, we define  $G$  to be a directed graph with a node corresponding to each glue and, for each distinct input assembly  $g_1[s]g_2$ , a directed edge from  $g_1$  to  $g_2$ . Then each distinct assembly in  $c$  corresponds to a source-sink pair in  $G$ , and each possible way to construct that assembly corresponds to a path in  $G$  from the source of the assembly to the sink of the assembly.

Say that there exist three glues  $g_1, g_2, g_3$  such that  $(g_1, g_2)$  and  $(g_2, g_3)$  are edges in  $G$  but  $(g_1, g_3)$  is not an edge in  $G$ . Then we can mix the assembly corresponding to the edge  $(g_1, g_2)$  with the assembly corresponding to the edge  $(g_2, g_3)$  to get an assembly with glue  $g_1$  to the west and glue  $g_3$  to the east. This is equivalent to adding the edge  $(g_1, g_3)$  to  $G$ . Each such mixing requires us to add a constant number of nodes and edges to the mix DAG  $B$ , and increases the number of edges in  $G$  by 1. The graph  $G$  can never have more than  $k^2$  edges, so repeated mixings of this type add a total of  $O(k^2)$  work to  $B$ . Hence, any mixing of bins in  $A$  can be replaced by  $O(k^2)$  binary mixes in  $B$ . As a result,  $|B|$  has size  $O(k^2|A|)$ , and can therefore be converted to an RCFG with size  $O(k^2|A|)$  by Theorem 2.  $\square$

**Theorem 7.** *With respect to the total number of distinct glue pairs  $k$ , the separation for any string is  $O(k^2)$ .*

*Proof.* Let  $A$  be a SAS using  $k$  glues that generates a string  $s$ . By Lemma 4, there is an RCFG of size  $O(k^2|A|)$  that generates  $s$ . So separation is at most  $O(k^2)$ .  $\square$

**Theorem 8.** *With respect to the length of the string  $n$ , the separation for any binary string is  $O((n/\log n)^{2/3})$ .*

*Proof.* Let  $A$  be a SAS generating a string  $s$  of length  $n$ . Let  $k$  be the number of glues used in  $A$ . Either  $k = O((n/\log n)^{1/3})$  or  $k = \omega((n/\log n)^{1/3})$ . If  $k = O((n/\log n)^{1/3})$  then by Lemma 4 there is an RCFG of size  $O(k^2|A|) = O((n/\log n)^{2/3} \cdot |A|)$  generating  $s$ . So the separation is at most  $O((n/\log n)^{2/3})$ . Now suppose  $k$  is  $\omega((n/\log n)^{1/3})$ . Then  $|A| = \omega((n/\log n)^{1/3})$ . Lemma 2 of Section 2.2 in [5] shows that there is an RCFG of size  $O(n/\log n)$  generating  $s$ . Hence, the separation is  $o((n/\log n)^{2/3})$ . So in both cases the separation is  $O((n/\log n)^{2/3})$ .  $\square$

**Acknowledgements.** We thank Martin Demaine, André Schulz, Diane Souvaine, and Hyunmin Yi for helpful discussions, and anonymous reviewers for helpful suggestions.

## References

1. Charikar, M., Lehman, E., Liu, D., Panigrahy, R., Prabhakaran, M., Rasala, A., Sahai, A., Shelat, A.: Approximating the smallest grammar: Kolmogorov complexity in natural models. In: Proceedings of the 34th Annual ACM Symposium on Theory of Computing, New York, NY, USA, pp. 792–801. ACM, New York (2002)
2. Demaine, E., Demaine, M., Fekete, S., Ishaque, M., Rafalin, E., Schweller, R., Souvaine, D.: Staged self-assembly: nanomanufacture of arbitrary shapes with  $O(1)$  glues. *Natural Computing* 7, 347–370 (2008)
3. Farach, M., Thorup, M.: String matching in Lempel-Ziv compressed strings. *Algorithmica* 20, 388–404 (1998)
4. Göös, M., Orponen, P.: Synthesizing minimal tile sets for patterned dna self-assembly. In: Sakakibara, Y., Mi, Y. (eds.) DNA 16 2010. LNCS, vol. 6518, pp. 71–82. Springer, Heidelberg (2011)
5. Lehman, E.: Approximation Algorithms for Grammar-Based Data Compression. PhD thesis, MIT (2002)
6. Ma, X., Lombardi, F.: Synthesis of tile sets for dna self-assembly. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27(5), 963–967 (2008)
7. Rytter, W.: Application of Lempel-Ziv Factorization to the Approximation of Grammar-Based Compression. In: Apostolico, A., Takeda, M. (eds.) CPM 2002. LNCS, vol. 2373, pp. 20–31. Springer, Heidelberg (2002), doi:10.1007/3-540-45452-7\_3.
8. Sakamoto, H.: A fully linear-time approximation algorithm for grammar-based compression. *Journal of Discrete Algorithms* 3(2-4), 416–430 (2005)
9. Soloveichik, D., Winfree, E.: Complexity of Self-assembled Shapes. In: Ferretti, C., Mauri, G., Zandron, C. (eds.) DNA 2004. LNCS, vol. 3384, pp. 344–354. Springer, Heidelberg (2005)
10. Winfree, E.: Algorithmic Self-Assembly of DNA. PhD thesis, Caltech (1998)

# Computing Maximal Kleene Closures That Are Embeddable in a Given Constrained DNA Language\*

Stavros Konstantinidis and Nicolae Santean

Department of Mathematics and Computing Science, Saint Mary's University,  
Halifax, Nova Scotia, B3H 3C3 Canada

{s.konstantinidis,nic.santean}@smu.ca

**Abstract.** We consider the problem of characterizing nontrivial languages  $D$  that are maximal with the property that  $D^*$ , the Kleene closure of  $D$ , is contained in the subword closure of a given set  $S$  of words of some fixed length  $k$ . The subword closure of  $S$  is simply the set of words for which all subwords of length  $k$  are in  $S$ . We provide a deep structural characterization of these languages  $D$ , which leads to polynomial time algorithms for computing such languages. This work is motivated by the problem of encoding arbitrary data into a set of DNA molecules such that all blocks of length  $k$  in these molecules satisfy the constraint  $S$  – eg, they can form no stable bonds between them, or they have a desired g-c ratio.

**Keywords:** algorithm, automaton, code, DNA encodings, maximal, regular language, subword closure.

## 1 Introduction

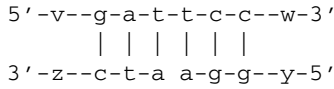
Given a set  $S$  of words of some fixed length  $k$ , the subword closure  $S^\circledast$  of  $S$  is the set of all words whose subwords of length  $k$  must be in  $S$ . This concept has been considered in the context of coding for reliable DNA computing – see [11], [4], [5] – as well as in state complexity considerations [1]. It was introduced in [11] to provide a structural characterization of all maximal bond-free languages, which were previously introduced in [10] as  $\theta$ - $k$ -codes. In these applications, the set  $S$  represents a subword constraint and the fact that some language is a subset of  $S^\circledast$  means that the language satisfies that constraint. In DNA computing, for instance, it is often desirable that no DNA molecules in the test tube contain two short blocks of  $k$  pairwise complementary<sup>1</sup> bases (nucleotides), as this would allow a sufficiently strong bond to form between molecules – see Fig. 1. In this case, the constraint  $S$  is a set of words of length  $k$  representing short molecules that are not pairwise complementary, and the language  $S^\circledast$  represents molecules (possibly arbitrarily long) such that there can be no sufficiently strong bonds between any two of them. For example, the constraint

$$S = \{aca, cac, caa, aac, aga, gaa, aag, agg, gga\} \quad (1)$$

---

\* Research supported by NSERC.

<sup>1</sup> Recall that there are four DNA bases, denoted as  $a, c, g, t$ , such that the Watson/Crick complement of  $a$  is  $t$ , and vice versa, and the complement of  $c$  is  $g$ , and vice versa. If we write  $\theta(\sigma)$  for the complement of the base  $\sigma$  then, for a DNA word  $\sigma_1 \cdots \sigma_n$ , the complement is  $\theta(\sigma_n) \cdots \theta(\sigma_1)$ . For example, the complement of  $agggt$  is  $acct$ .



**Fig. 1.** Two DNA molecules:  $5'-vgattccw-3'$  and  $5'-yggaatcz-3'$ . The symbols  $a, c, g, t$  represent the four DNA bases (nucleotides), and  $v, w, y, z$  are arbitrary sequences of DNA bases. Vertical bars represent chemical bonds between complementary nucleotides. The complementary parts are  $5'-gattcc-3'$  and  $5'-ggaatc-3'$ . In terms of language theory, the words  $vgattccw$  and  $yggaatcz$  contain two complementary subwords  $gattcc$  and  $ggaatc$  of length  $k = 6$ .

is such that no two words in  $S$  are complementary – for example, the complement  $tgt$  of  $aca$  is not equal to any word in  $S$ . Then, the subword closure  $S^\otimes$  contains arbitrarily long words such that these words contain no two subwords of length at least three that are complementary. For example, the word  $cac(aca)^n$  is in  $S^\otimes$ , for all positive integers  $n$ . The relevance of languages  $S^\otimes$  in avoiding unwanted hybridizations during DNA computations has been tested successfully in practice [13].

In [4] and [5], the authors consider the problem of encoding arbitrary sequences of data blocks into the words of  $S^\otimes$ , for a given constraint  $S$  of some length  $k$ . More specifically, if  $\Delta^m$  is the set of all words of length  $m$  over some alphabet  $\Delta$ , for some  $m \geq 1$ , the authors propose and implement a method for encoding  $\Delta^m$  onto some set of words  $D$  with  $D^* \subseteq S^\otimes$ . In practice,  $\Delta$  could be any high level alphabet, and  $S^\otimes$  could be a bond-free DNA language (over the alphabet  $\Sigma = \{a, c, g, t\}$ ). Then, the method ensures that any data sequence  $v_1 v_2 v_3 \dots$ , with  $v_i \in \Delta^m$ , can be encoded onto a word  $w_1 w_2 w_3 \dots$  in  $S^\otimes$ , which corresponds to a DNA molecule. Then, no two such molecules in the test tube would form a sufficiently strong set of bonds. Another example of  $S$  could be a set of words with a certain desirable  $g-c$  ratio. In this case, for any arbitrarily long  $w$  in  $S^\otimes$ , if  $v$  is a subword of  $w$  of length  $k$  then  $v$  has the right  $g-c$  ratio.

In this paper, motivated by the above problems, but *independently* of the meaning of the constraint  $S$ , we consider the problem of characterizing *nontrivial* languages  $D$  whose words are of length at least  $k$  and are *maximal* with the property “ $D^* \subseteq S^\otimes$ ” – the nontrivial requirement for  $D$  ensures that  $D^*$  is of exponential density and has a good encoding capability. We obtain a *complete* structural characterization of these languages  $D$ , which leads to polynomial time algorithms for computing such languages, and a better understanding of the encoding method in [4] and [5]. We note that the more general question of computing maximal regular languages  $D$  such that  $D^* \subseteq R$ , where  $R$  is any regular language, has been solved recently in [12] using different tools. However, these tools lead to an algorithm with an exponential number of steps.

Consider, for instance, the constraint  $S$  in (1), and assume for the sake of the example that this constraint captures all requirements for reliable storage of data in the form of single DNA strands. Depending on the choice made by a certain nondeterministic step, our algorithm is capable of computing the following maximal  $D$ 's such that  $D^* \subseteq S^\otimes$

$$\begin{aligned}
 D_1 &= caa((ca)^+ a)^* (ca)^* + cac(a(aca)^* c)^* a(aca)^* (\lambda + a) \\
 D_2 &= aga(a(ggaa)^* ga)^* (\lambda + a(ggaa)^* gga) + agg((aag)^+ g)^* (aag)^* a,
 \end{aligned}$$

where we have used notation of regular expressions for the languages  $D_1$  and  $D_2$ . In fact, as customary with regular language algorithms, those  $D$ 's are represented in our algorithms as finite automata and, then, one can easily compute, for any given  $\ell$ , the set  $D(\ell)$  of all words in  $D$  of length  $\ell$ . If we need to encode sequences of  $n$  different objects, we can pick an  $\ell$  such that  $D(\ell)$  contains at least  $n$  words  $w_1, \dots, w_n$  of length  $\ell$ . Then, any data sequence is encoded into a DNA word of the form  $w_{i_1} w_{i_2} \dots$  which is in  $D(\ell)^*$  and, hence, also in  $S^\otimes$ .

To take the above example further, assume that we want to store an 8-bit colour image of size  $1024 \times 512$  pixels into  $S^\otimes$ . One way to do this is to pick an appropriate subset<sup>2</sup>  $D_1(\ell)$  of  $S^\otimes$ , and to define 1024 single DNA strands  $w_0, w_1, \dots, w_{1023}$ , with each one corresponding to one pixel row of the image. More specifically, each  $w_i$  is of the form

$$w_i = w_{i,0} w_{i,1} \dots w_{i,512}$$

such that each  $w_{i,j}$  is in  $D_1(\ell)$ , the word  $w_{i,0}$  encodes the row number (between 0 and 511), and each  $w_{i,j}$  with  $j \geq 1$  encodes the colour of the pixel  $(i, j - 1)$ .

In general, there might be several maximal  $D$ 's such that  $D^* \subseteq S^\otimes$ . The problem of computing "good" such  $D$ 's according to some criteria – e.g., large number of words in  $D$  of some given length – is not addressed here, as it requires further research. As an example, we note that the language  $D_1$  above contains nine words of length 11, and  $D_2$  contains only three words of length 11.

The paper is organized as follows. Section 2 contains the basic notation and terminology about regular languages, automata and the subword closure operation. Section 3 deals with characterizing structurally nontrivial languages  $D$  whose words are of length at least  $k$  and are maximal with " $D^* \subseteq S^\otimes$ ". Our characterization is used in Section 4 to evaluate the encoding method of [4] and [5]. Moreover, this characterization is used in Section 5 to obtain polynomial algorithms for constructing certain maximal languages  $D$ . Finally, Section 6 contains a few concluding remarks and suggestions for future research.

## 2 Basic Notation and Background

This section uses [14,16,15] as general references.

### 2.1 Words, Languages, Codes

For a set  $S$ , we denote by  $|S|$  the cardinality of  $S$ . We consider an arbitrary alphabet  $\Sigma$  containing at least two symbols. The expressions  $\Sigma^*$ ,  $\lambda$ ,  $\Sigma^+$ ,  $|w|$  denote, respectively, the set of all words over  $\Sigma$ , the empty word, the set of all nonempty words, and the length of a word  $w$ . For an integer  $n \geq 0$ ,  $(w)^n$  is the word consisting of  $n$  copies of  $w$ . A *prefix* (resp. *suffix*, *subword*) of a word  $w$  is any word  $u$  such that  $w = ux$  (resp.  $w = xu$ ,  $w = xuy$ ) for some words  $x, y$ . A subword of  $w$  is also called a factor, or infix, of  $w$ . A language is any set of words. A word  $w$  is called an  $L$ -word if  $w \in L$ . As usual, for any integer  $n \geq 0$ , if  $L$  is a language then  $L^n$  is the language whose words consist

<sup>2</sup> In this case,  $\ell$  should be such that  $D_1(\ell)$  contains at least 512 elements.



of any  $n$  concatenated words from  $L$ . Also,  $L^*$  is the union of  $L^n$ , for all  $n \geq 0$ , and  $L^+ = L^* - \{\lambda\}$ . For any word  $x$  and language  $L$  we use the notation  $x^{-1}L = \{z \in \Sigma^* \mid xz \in L\}$ . In particular, if  $x$  is a prefix of some word  $w$ , then  $x^{-1}w$  is the suffix  $z$  of  $w$  such that  $w = xz$ . A language  $C$  is called a (uniquely decodable) *code* if, for every word  $w \in C^+$ , there is exactly one sequence of  $C$ -words whose concatenation is equal to  $w$ . Any language whose words are of some fixed length is always a code (usually called a uniform, or block, code). A language  $D$  is called *nontrivial* if it contains two words  $w_1, w_2$  such that  $w_1w_2 \neq w_2w_1$ . Note that, in this case, the set  $\{w_1w_2, w_2w_1\}$  is a two-element block code, and is a subset of  $D^*$ . A language  $L$  is called *maximal with respect to some property ‘ $\mathcal{P}$ ’*, if any language  $L'$  containing  $L$  and satisfying ‘ $\mathcal{P}$ ’ is equal to  $L$ .

## 2.2 Automata, Graphs, Cycles

A complete deterministic finite automaton is a quintuple  $M = (\Sigma, K, \delta, s, F)$  such that  $K$  is the state set,  $s$  is the start state,  $F$  is the set of final states and  $\delta : K \times \Sigma \rightarrow K$  is the transition function, which is extended as  $\delta : K \times \Sigma^* \rightarrow K$  in the usual way. If  $\delta$  is partial then  $M$  is not complete. In any case, we call it a *DFA*. A triple  $(p, \sigma, q)$  with  $\sigma \in \Sigma$  and  $\delta(p, \sigma) = q$  is called a *transition* of  $M$ . In this case, we say that the transition is going out of state  $p$ . The DFA  $M$  can be viewed as a directed labeled graph. A *path* of  $M$  is a sequence  $(p_0, \sigma_1, p_1, \dots, \sigma_n, p_n)$  such that  $(p_{i-1}, \sigma_i, p_i)$  is a transition of  $M$ , for all  $i = 1, \dots, n$ . In this case, the word  $\sigma_1 \dots \sigma_n$  is called the *label* of the path. The path is *accepting* if  $p_0$  is the start state and  $p_n$  is a final state. The language  $L(M)$  *accepted* by  $M$  is the set of labels in all accepting paths of  $M$ . These languages constitute the class of *regular languages* – see [16,15] for more information on regular languages.

The DFA  $M$  is called *trim* if every state of  $M$  occurs in some accepting path of  $M$ . The *size* of  $M$  is  $|K| + |T|$ , that is the number of states plus the number of transitions in  $M$ . We note that if  $M$  is trim then  $|K| \leq |T| + 1$  and, therefore, the size of  $M$  is dominated by  $|T|$ . A state in an automaton is called a *fork state* if there at least two transitions going out of that state. A *cycle* in the DFA is a path in which the first and last states of the path are equal. The special cycle  $(p)$ , where  $p$  is any state, is called *trivial*. A *strongly connected component (SCC)*, with respect to a DFA  $M$ , is a set  $\mathcal{C}$  of states that is maximal with the property that there is a path in  $M$  between any pair of states in  $\mathcal{C}$ . The component  $\mathcal{C}$  is called *nontrivial* if there is at least one transition between some states in  $\mathcal{C}$ . For the sake of simplicity, we shall say that a component  $\mathcal{C}$  *contains* a transition (or a path) to mean that the DFA in which  $\mathcal{C}$  exists contains that transition (or path) with all states involved belonging to  $\mathcal{C}$ .

## 2.3 The Subword Closure $S^\otimes$ and the DFA $\text{Trie}(S)^\otimes$

As mentioned before, any nonempty set  $S$  of words of length  $k$ , for some integer  $k > 0$ , is called a *subword constraint*. It is used to define the language

$$S^\otimes \triangleq \{w \in \Sigma^* \mid \text{if } u \text{ is a subword of } w \text{ and } |u| = k \text{ then } u \in S\}.$$

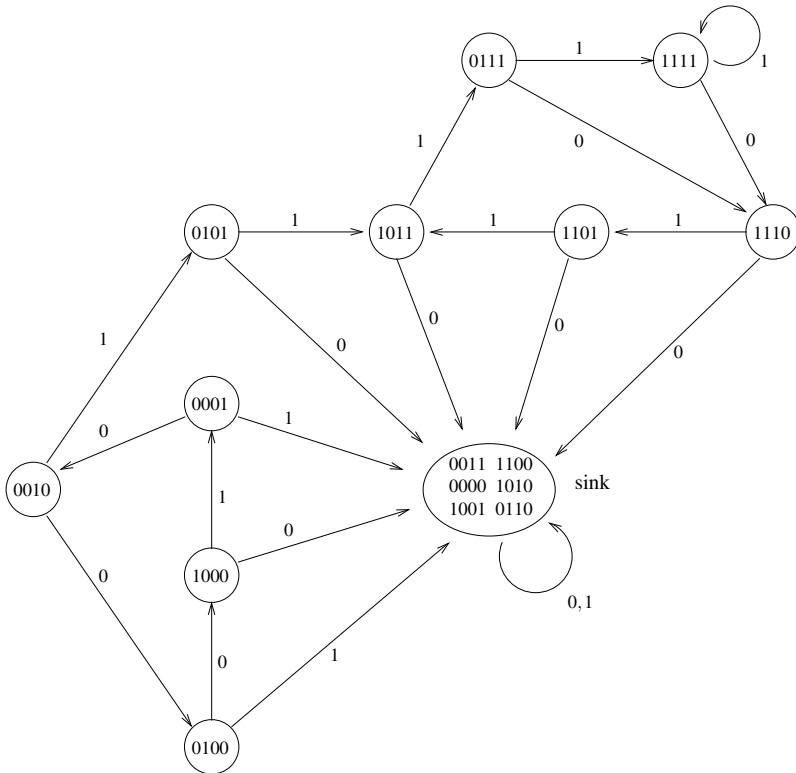
Two properties of  $S^\otimes$  are the following: (1) If  $w \in S^\otimes$  then every subword of  $w$  is in  $S^\otimes$ . (2) If  $xu, vy \in S^\otimes$  and  $|u| = |v| = k$ , then  $xuvy \in S^\otimes$  if and only if  $uv \in S^\otimes$ .

In [11] it is shown that every language  $S^\otimes$  is regular via the DFA  $\text{Trie}(S)^\otimes$ , which is defined as follows. First, let  $\text{Trie}(S)$  be the *trie* accepting the set  $S$ . Recall [3] this is the complete DFA with states  $\{[u] \mid u \text{ is a prefix of a word in } S\} \cup \{[\text{sink}]\}$  such that  $[\lambda]$  is the start state and  $\{[u] \mid u \in S\}$  is the set of final states. We remark that the notation  $[\cdot]$  for states is only used to help the reader distinguish easily that  $u$  represents a word and  $[u]$  represents a state. By extending this notation to sets of states, we can write that the set of final states of  $\text{Trie}(S)$  is  $[S]$ . The transition function  $\delta$  of  $\text{Trie}(S)$  is such that  $\delta([u], \sigma) = [u\sigma]$ , when  $u\sigma$  is a prefix of  $S$  of length at most  $k$ , and with all the other values of  $\delta$  being  $[\text{sink}]$ .

The DFA  $\text{Trie}(S)^\otimes$  accepting  $S^\otimes$  is obtained from the trie  $\text{Trie}(S)$  as follows [11] – see also Fig. 2. The set of states is the same; the start state is the same; all states now are final; the transition function  $\delta^\otimes$  of  $\text{Trie}(S)^\otimes$  is the same as  $\delta$  except as follows: for each  $u \in S$  and  $\sigma \in \Sigma$ , if  $u \in \Sigma u_1$  and  $u_1\sigma \in S$ , then  $\delta^\otimes([u], \sigma) = [u_1\sigma]$  – this ensures that the last  $k$  symbols read drive the automaton to a state in  $[S]$ .

*Remark 1.* A few useful properties of  $\text{Trie}(S)^\otimes$  are the following.

- If  $([u], \sigma_1, p_1, \dots, \sigma_k, p_k)$  is a path in  $\text{Trie}(S)^\otimes$  and  $p_k \neq [\text{sink}]$  then the state  $p_k$  must be  $[\sigma_1 \dots \sigma_k]$ .



**Fig. 2.** The part of  $\text{Trie}(S)^\otimes$  involving only states  $[u]$ , with  $u \in S$ , and the state  $[\text{sink}]$

- If  $w \in S^\otimes$  and  $|w| \geq k$  then  $\delta^\otimes([\lambda], w) = \delta^\otimes([x], w_1) = [y]$ , where  $x$  is the prefix of  $w$  of length  $k$ ,  $w_1 = x^{-1}w$ , and  $y$  is the suffix of  $w$  of length  $k$ .
- The DFA  $\text{Trie}(S)^\otimes$  can be computed in linear time with respect to the size of  $S$  – this size is the sum of the lengths of all words in that set.
- Any nontrivial SCC  $[Q]$  of  $\text{Trie}(S)^\otimes$  is such that  $Q \subseteq S$ .

Fig. 2 shows a part of the DFA  $\text{Trie}(S)^\otimes$  accepting the language  $S^\otimes$ , where

$$S = \{0001, 0010, 0100, 0101, 0111, 1000, 1011, 1101, 1110, 1111\}. \quad (2)$$

For simplicity, in this example, we used the alphabet  $\{0, 1\}$ .

### 3 Characterizing Maximal $D$ 's with $D^* \subseteq S^\otimes$

In this section we fix an arbitrary subword constraint  $S$  of some length  $k$ , that is, a nonempty language  $S \subseteq \Sigma^*$  with words of length  $k > 0$ . The first main problem is to characterize structurally any *nonempty* language  $D$  whose words are of length at least  $k$  and is maximal with the property “ $D^* \subseteq S^\otimes$ ”. It turns out (see Theorem 1) that, for such a  $D$ , there is a nontrivial SCC  $[Q]$  of  $\text{Trie}(S)^\otimes$ , with  $Q \subseteq S$ , such that for all  $D$ -words, their first  $k$  symbols drive the DFA  $\text{Trie}(S)^\otimes$  to a certain set of states  $[X] \subseteq [Q]$ , the  $D$ -words get accepted at a set of states  $[Y] \subseteq [Q]$  that depends on  $X$  and is denoted as  $[Y] = [Q_X^\triangleright]$ , and  $Y$  has the property that, from any state  $[y] \in [Y]$  and on input  $x$ , the automaton  $\text{Trie}(S)^\otimes$  gets to the state  $[x]$ , for any  $x \in X$  – see Definition 1 and Fig. 3.

The second main problem is to characterize structurally any *nontrivial* language  $D$  whose words are of length at least  $k$  and is maximal with the property “ $D^* \subseteq S^\otimes$ ”. The characterization is the same as the one for a *nonempty*  $D$  with the additional requirement that the strongly connected component  $[Q]$  contains a fork state – see Theorem 2. The fact that  $D$  is nontrivial allows one to encode in  $D^*$  – therefore also in  $S^\otimes$  – arbitrary data. Indeed,  $D$  being nontrivial means that it contains words  $w_1, w_2$  with  $w_1 w_2 \neq w_2 w_1$ , which implies that  $\{w_1 w_2, w_2 w_1\}^* \subseteq D^* \subseteq S^\otimes$ . Then we can encode into  $S^\otimes$  arbitrary sequences from a set of data blocks  $\{v_1, \dots, v_n\}$  as follows. Let

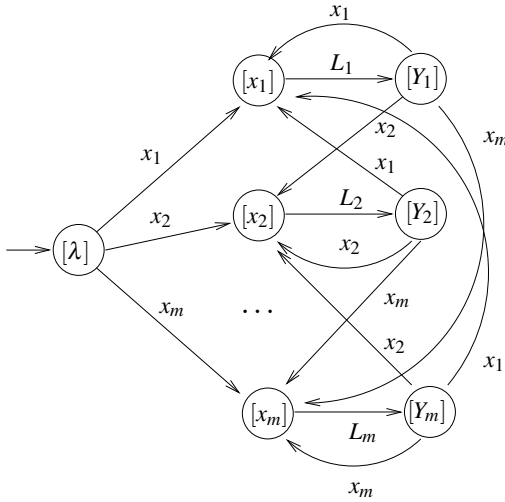
$$C_n = \{w_1 w_2, w_2 w_1\}^{\lceil \log n \rceil}.$$

Each  $v_i$  is encoded using a unique element of  $C_n$ . As  $C_n^* \subseteq D^*$ , we can encode arbitrary sequences of  $v_i$ 's into  $S^\otimes$ , satisfying thus the constraint  $S$ . Typically, the set of data blocks could be all words of length  $m$  over a certain data alphabet, where  $m$  is some positive integer.

**Definition 1.** Let  $Q$  be a nonempty subset of  $S$  such that  $[Q]$  is a nontrivial strongly connected component of  $\text{Trie}(S)^\otimes$ . For any nonempty subset  $X$  of  $Q$ , we define

$$Q_X^\triangleright \triangleq \{v \in Q \mid \forall x \in X : \delta^\otimes([v], x) = [x]\} = \{v \in Q \mid \forall x \in X : vx \in S^\otimes\}.$$

Thus,  $[Q_X^\triangleright]$  is the set of states in  $[Q]$  for which any input word  $x \in X$  drives the automaton  $\text{Trie}(S)^\otimes$  to the state  $[x]$ .



**Fig. 3.** The sets of states  $[X] = \{[x_1], \dots, [x_m]\}$  and  $[Y] = [Q_X^\triangleright] = [Y_1] \cup \dots \cup [Y_m]$  are subsets of some strongly connected component  $[Q]$  of  $\text{Trie}(S)^\otimes$ . Each node  $[Y_i]$  in the diagram represents one or more states in  $[Y]$ , and each language  $L_i$  consists of the labels of all paths from  $[x_i]$  to  $[Y_i]$ . Then, the language  $\langle Q, X \rangle$  is equal to  $\cup_{i=1}^m x_i L_i$ .

*Example 1.* In Fig. 2, let  $Q = \{0111, 1111, 1110, 1101, 1011\}$ . Then,

$$Q_{\{1011\}}^\triangleright = \{1011, 0111, 1111\}$$

and, for  $X = \{0111, 1011\}$ , we have that

$$Q_X^\triangleright = \{0111, 1111\}.$$

The major structural observation for the desired languages  $D$  is that they can be expressed in terms of the sets

$$\langle Q, X \rangle_x \triangleq \{w \in \Sigma^* \mid \delta^\otimes([x], w) \in [Q_X^\triangleright]\}$$

for all  $x \in X$ , where  $[Q]$  is a strongly connected component of  $\text{Trie}(S)^\otimes$  – thus,  $Q \subseteq S$  – and  $X \subseteq Q$ . Let

$$\langle Q, X \rangle \triangleq \bigcup_{x \in X} x \langle Q, X \rangle_x.$$

Obviously, the words of this language are of length at least  $k$ . This notation is also important in the section on algorithmic considerations.

*Example 2.* In Fig. 2, for  $Q = \{0111, 1111, 1110, 1101, 1011\}$  and  $X = \{0111, 1011\}$ , we have that

$$\begin{aligned} \langle Q, X \rangle_{1011} &= 1(\lambda + 11^*)(0111(\lambda + 11^*))^* \text{ and} \\ \langle Q, X \rangle_{0111} &= (\lambda + 11^*)(0111(\lambda + 11^*))^*, \end{aligned}$$

where we have used notation of regular expressions for denoting languages.

**Theorem 1.** *Let  $S$  be any subword constraint of some length  $k$ , and let  $D$  be any nonempty language whose words are of length at least  $k$ . Then,  $D$  is maximal with  $D^* \subseteq S^\otimes$  if and only if there are nonempty subsets  $X, Y, Q$  of  $S$  such that*

$$D = \langle Q, X \rangle = S^\otimes \cap X\Sigma^* \cap \Sigma^*Y,$$

and  $X, Y \subseteq Q$ ,  $[Q]$  is a nontrivial strongly connected component of  $\text{Trie}(S)^\otimes$ ,  $Y = Q_X^>$ , and  $X$  is maximal with “ $X \subseteq Q$  and  $Q_X^> = Y$ ”.

**Theorem 2.** *Let  $S$  be any subword constraint of some length  $k$ , and let  $D$  be any nontrivial language whose words are of length at least  $k$ . Then,  $D$  is maximal with  $D^* \subseteq S^\otimes$  if and only if there are nonempty subsets  $X, Y, Q$  of  $S$  such that*

$$D = \langle Q, X \rangle = S^\otimes \cap X\Sigma^* \cap \Sigma^*Y,$$

and  $X, Y \subseteq Q$ ,  $[Q]$  is a nontrivial strongly connected component of  $\text{Trie}(S)^\otimes$  containing a fork state,  $Y = Q_X^>$ , and  $X$  is maximal with “ $X \subseteq Q$  and  $Q_X^> = Y$ ”.

The proofs of the above results rely on a sequence of technical lemmata. The first one gives a taste of what it means when  $D^* \subseteq S^\otimes$ , without necessarily requiring that  $D$  is maximal with this property.

**Lemma 1.** *Let  $D$  be a nonempty language whose words are of length at least  $k$ .*

1. *If  $D^* \subseteq S^\otimes$  then  $D = \bigcup_{x \in X} x(x^{-1}D)$  and  $x(x^{-1}D)y \subseteq S^\otimes$ , for all  $x, y \in X$ , where  $X$  is the set of all prefixes of  $D$  of length  $k$ .*
2. *If there is a subset  $X$  of  $S$  and languages  $D_x$ , for all  $x \in X$ , such that  $D = \bigcup_{x \in X} (xD_x)$  and  $xD_xy \subseteq S^\otimes$ , for all  $x, y \in X$ , then  $D^* \subseteq S^\otimes$ .*

**Lemma 2.** *Let  $X, Q$  be nonempty subsets of  $S$  such that  $X \subseteq Q$  and  $[Q]$  is a nontrivial strongly connected component of  $\text{Trie}(S)^\otimes$ .*

1.  $\langle Q, X \rangle^* \subseteq S^\otimes$ .
2.  $\langle Q, X \rangle = S^\otimes \cap X\Sigma^* \cap \Sigma^*Q_X^>$ .
3. *If  $Q_X^> \neq \emptyset$  then, for all  $x \in X$ , we have that  $\langle Q, X \rangle_x \neq \emptyset$ .*

Now we present the next major step to proving the desired theorem. We establish that any  $D$  with  $D^* \subseteq S^\otimes$  must be a subset of some language of the form  $\langle Q, X \rangle$ . This result allows us to focus on a strongly connected component of  $\text{Trie}(S)^\otimes$  when we wish to characterize structurally those  $D$ 's.

**Lemma 3.** *If  $D$  is a nonempty language whose words are of length at least  $k$  and  $D^* \subseteq S^\otimes$ , then*

$$D \subseteq \langle Q, X \rangle,$$

for some nonempty subsets  $Q, X$  of  $S$  with  $X \subseteq Q$  and  $[Q]$  a nontrivial strongly connected component of  $\text{Trie}(S)^\otimes$ .

The last technical lemma before the proof of Theorem 1 deals with containment relationships between sets of the form  $Q_X^>$  and  $P_Z^>$ .

**Lemma 4.** *Let  $X, Z, Q, P$  be nonempty subsets of  $S$  such that  $X \subseteq Q$ ,  $Z \subseteq P$ ,  $[Q]$  and  $[P]$  are nontrivial strongly connected components of  $\text{Trie}(S)^\otimes$ , and  $Q_X^\succ \neq \emptyset$ .*

1. *If  $\langle Q, X \rangle \subseteq \langle P, Z \rangle$  then  $X \subseteq Z$ .*
2. *If  $Z \subseteq Q$  and  $\langle Q, X \rangle = \langle Q, Z \rangle$  then  $X = Z$ .*

Proof of Theorem 1. First we do the ‘only if’ part. So suppose that  $D$  is maximal with  $D^* \subseteq S^\otimes$ . Then,  $D \subseteq \langle Q, X \rangle$  according to Lemma 3. At the same time, Lemma 2 says that  $\langle Q, X \rangle^* \subseteq S^\otimes$ . As  $D$  is maximal we have that, in fact,  $D = \langle Q, X \rangle$ . Let  $Y = Q_X^\succ$ . By Lemma 2, we have  $D = S^\otimes \cap X\Sigma^* \cap \Sigma^*Y$ . As  $D$  is nonempty, we have that  $Y$  is nonempty as well.

It remains to show that  $X$  is maximal with “ $X \subseteq Q$  and  $Q_X^\succ = Y$ ”. So suppose that  $X \subseteq Z \subseteq Q$  and  $Q_Z^\succ = Y$ . Then, we need to show that  $Z = X$ . For this it suffices to show that  $\langle Q, X \rangle = \langle Q, Z \rangle$ . In turn, this would follow by the maximality of  $D$  if we show that  $D \subseteq \langle Q, Z \rangle$ . So take any  $w \in D = \langle Q, X \rangle$ . Then,  $w = xw_1$  for some  $x \in X$  and  $w_1 \in \langle Q, X \rangle_x$ , which implies  $\delta^\otimes([x], w_1) \in [Q_X^\succ] = [Q_Z^\succ]$ . Also, as  $x \in Z$  we have that  $w_1 \in \langle Q, Z \rangle_x$  and, therefore  $xw_1 \in \langle Q, Z \rangle$ , as required.

Now we do the ‘if’ part. By Lemma 2, we have  $D^* \subseteq S^\otimes$ . To show that  $D$  is maximal, we assume that  $D \subseteq B$  and  $B^* \subseteq S^\otimes$ , for some language  $B$ , and we deduce that  $B = D$ . By Lemma 3, we have  $B \subseteq \langle P, Z \rangle$ , where  $Z, P$  are nonempty subsets of  $S$ ,  $Z \subseteq P$ , and  $[P]$  is a nontrivial strongly connected component of  $\text{Trie}(S)^\otimes$ . This implies that  $\langle Q, X \rangle \subseteq \langle P, Z \rangle$ . It suffices to show that  $P = Q$  and  $X = Z$ . By Lemma 4, we get  $X \subseteq Z$ , so there is a state belonging to both  $[Q]$  and  $[P]$ . This implies  $P = Q$ . Also, obviously  $Q_Z^\succ = P_Z^\succ$ . As  $X$  is maximal with “ $X \subseteq Q$  and  $Q_X^\succ = Y$ ” and  $X \subseteq Z \subseteq Q$ , it suffices to show that  $Q_Z^\succ = Q_X^\succ$ .

First, by definition of  $Q_X^\succ$ ,  $X \subseteq Z$  implies  $Q_Z^\succ \subseteq Q_X^\succ$ . For the converse inclusion, take any  $v \in Q_X^\succ$ . Also, take any  $x \in X$ . As  $x, v \in Q$ , there is a path from  $[x]$  to  $[v]$  having some label  $w$ , which implies  $w \in \langle Q, X \rangle_x$ . As  $\delta^\otimes([\lambda], xw) = [v]$ , there is a word  $w'$  such that  $xw = w'v$ . So  $w'v \in \langle Q, X \rangle$  and, therefore,  $w'v \in \langle Q, Z \rangle$ . Then, by Lemma 2, we have  $w'v \in \Sigma^*Q_Z^\succ$ ; hence,  $v \in Q_Z^\succ$ , as required.  $\square$

The next two lemmata are required for the proof of Theorem 2, which involves a fork state in the strongly connected component  $[Q]$ . In particular, if  $[v]$  is a fork state with transitions  $([v], \sigma_1, [x_1])$  and  $([v], \sigma_2, [x_2])$ , then there is a path of length  $k - 1$  from some state  $[u]$  to the fork state  $[v]$ . Moreover, the labels of the paths from  $[u]$  to  $[x_1]$  and  $[x_2]$  are  $x_1$  and  $x_2$ , and it turns out that  $\langle Q, \{x_1, x_2\} \rangle$  is a nontrivial language.

**Lemma 5.** *If  $[Q]$  is a nontrivial strongly connected component of  $\text{Trie}(S)^\otimes$  then, for every  $v \in Q$  and  $n \geq 1$ , there is  $u \in Q$  and a path of length  $n$  from  $[u]$  to  $[v]$ .*

**Lemma 6.** *Let  $[Q]$  be a nontrivial strongly connected component of  $\text{Trie}(S)^\otimes$ .*

1. *If  $[Q]$  contains a fork state  $[v]$  having transitions to some distinct states  $[x_1], [x_2] \in [Q]$ , then  $\langle Q, X \rangle$  is a nontrivial language, where  $X = \{x_1, x_2\}$ .*
2. *There is a subset  $X$  of  $Q$  such that  $\langle Q, X \rangle$  is nontrivial if and only if  $[Q]$  contains a fork state.*

Proof of Theorem 2. The ‘if’ part is simply a weaker form of the ‘if’ part in Theorem 1. For the ‘only if’ part, we first apply Theorem 1: there are nonempty subsets  $X, Y, Q$  of

$S$  such that  $D = \langle Q, X \rangle = S^\otimes \cap X \Sigma^* \cap \Sigma^* Y$ , and  $X, Y \subseteq Q$ ,  $[Q]$  is a nontrivial strongly connected component of  $\text{Trie}(S)^\otimes$ ,  $Y = Q_X^\succ$ , and  $X$  is maximal with “ $X \subseteq Q$  and  $Q_X^\succ = Y$ ”. It remains to show that  $[Q]$  contains a fork state. But this follows immediately from Lemma 6.  $\square$

## 4 Connection with a Previous Method

In [4] and [5], in the context of encoding data into  $S^\otimes$ , the authors consider the problem of constructing a nonempty set  $B$  such that  $B^* \subseteq S^\otimes$ , using the following method.

1. Pick any nonempty subset  $Y$  of  $S$ .
2. Let  $S_Y = \{v \in S \mid \forall y \in Y : yv \in S^\otimes\}$ .
3. Let  $B_Y = S^\otimes \cap S_Y \Sigma^* \cap \Sigma^* Y$ .

In that method,  $S_Y$  is the set of possible  $S$ -words that can be appended to any  $Y$ -word without violating the constraint  $S$ . As expected, it can be shown that  $B_Y^* \subseteq S^\otimes$ . However, if we use a bad choice for  $Y$  then  $S_Y$  could be empty. Here we can evaluate the above method using the tools developed in the previous section. Clearly the set  $Y$  should be a subset of some  $Q$  such that  $[Q]$  is a strongly connected component of  $\text{Trie}(S)^\otimes$ . We define the following analogue of  $Q_X^\succ$ :

$$Q_Y^\prec \triangleq \{v \in Q \mid \forall y \in Y : \delta^\otimes([y], v) = [v]\} = \{v \in Q \mid \forall y \in Y : yv \in S^\otimes\}.$$

Then, the above set  $B_Y$  can be written as  $B_Y = S^\otimes \cap Q_Y^\prec \Sigma^* \cap \Sigma^* Y$ . As expected there is a strong connection between  $B_Y$  and  $\langle Q, X \rangle$ , where  $X = Q_Y^\prec$ .

**Lemma 7.** *Let  $X, Y, Q$  be nonempty subsets of  $S$  such that  $X, Y \subseteq Q$  and  $[Q]$  is a non-trivial strongly connected component of  $\text{Trie}(S)^\otimes$ .*

1. If  $Q_X^\succ = Y$  then  $X \subseteq Q_Y^\prec$ .
2. If  $Q_Y^\prec = X$  then  $Y \subseteq Q_X^\succ$ .
3. If  $X$  is maximal with “ $X \subseteq Q$  and  $Q_X^\succ = Y$ ” then  $X = Q_Y^\prec$ .
4. If  $Y$  is maximal with “ $Y \subseteq Q$  and  $Q_Y^\prec = X$ ” then  $Y = Q_X^\succ$ .
5.  $X$  is maximal with “ $X \subseteq Q$  and  $Q_X^\succ = Y$ ” if and only if  $Y$  is maximal with “ $Y \subseteq Q$  and  $Q_Y^\prec = X$ ”.

Thus, in the method of [4] and [5] with the requirement that  $Y \subseteq Q$ , the constructed set  $B_Y = S^\otimes \cap Q_Y^\prec \Sigma^* \cap \Sigma^* Y$  has the following properties.

- As  $Y \subseteq Q_X^\succ$ , we have  $B_Y \subseteq S^\otimes \cap X \Sigma^* \cap \Sigma^* Q_X^\succ = \langle Q, X \rangle$ , where  $X = Q_Y^\prec$ .
- If  $Y$  is chosen to be maximal with “ $Y \subseteq Q$  and  $Q_Y^\prec = X$ ,” then  $Y = Q_X^\succ$ ,

$$B_Y = S^\otimes \cap X \Sigma^* \cap \Sigma^* Q_X^\succ$$

and,  $X$  is maximal with “ $X \subseteq Q$  and  $Q_X^\succ = Y$ ”. Therefore,  $B_Y$  would be maximal with  $B_Y^* \subseteq S^\otimes$ .

## 5 Algorithmic Considerations for Maximal $D$ 's with $D^* \subseteq S^\otimes$

In this section we consider algorithms for the following problems.

- (P1) Given a subword constraint  $S$ , compute a DFA accepting a *nonempty* language  $D$  that is maximal with  $D^* \subseteq S^\otimes$ .  
 (P2) Given a subword constraint  $S$ , compute a DFA accepting a *nontrivial* language  $D$  that is maximal with  $D^* \subseteq S^\otimes$ .

We deal with the above problems by considering the following subproblems, which refer to a given  $\text{Trie}(S)^\otimes$  and a given nontrivial SCC  $[Q]$  of  $\text{Trie}(S)^\otimes$ .

- (SP1) Given a nonempty subset  $X$  of  $Q$ , compute the set  $Q_X^\succ$ .  
 (SP2) Given nonempty subsets  $Z, Y$  of  $Q$  such that  $Q_Z^\succ = Y$ , compute  $X$  such that  $Z \subseteq X$  and  $X$  is maximal with “ $X \subseteq Q$  and  $Q_X^\succ = Y$ ”.  
 (SP3) Given nonempty subsets  $X, Y$  of  $Q$  compute a DFA accepting  $S^\otimes \cap X \Sigma^* \cap \Sigma^* Y$ .

We shall use the abbreviation  $T$  for  $\text{Trie}(S)^\otimes$ . We use a bijective encoding of  $Q$  onto the set  $\bar{Q} = \{0, 1, \dots, |Q| - 1\}$ , such that, for  $v \in Q$ ,  $\bar{v}$  is the code of  $v$  in  $\bar{Q}$ . Using hashing techniques the encoding and decoding functions can be done in time  $O(1)$ . As customary, we realize any subset  $\bar{Z}$  of  $\bar{Q}$  as a Boolean array of size  $|Q|$  such that, for any  $z \in Q$ , we have that  $z \in Z$  if and only if the entry  $\bar{z}$  of the array is true. Thus, testing for membership in  $Z$  takes time  $O(1)$ .

### 5.1 Algorithm ASP1( $T, Q, X$ ) for (SP1), and the 2D Array BQ

A simple algorithm is to take any pair  $v \in Q$  and  $x \in X$ , and test whether  $\delta^\otimes([v], x) \neq [\text{sink}]$ . If, for the current  $v$ , the test is true for all  $x \in X$ , then  $v$  is added in  $Q_X^\succ$ . This algorithm performs in time  $O(|Q||X|k)$  and space  $O(|Q|)$ .

It turns out, however, that subproblem (SP1) needs to be solved repeatedly when we are looking for maximal solutions in the original main problems. For this reason, we shall need as a preprocessing step to compute a  $|Q| \times |Q|$  Boolean array BQ such that  $\text{BQ}[\bar{v}, \bar{v}']$  is true if and only if  $\delta^\otimes([v], v') \neq [\text{sink}]$ . This array can be computed in time  $O(|Q|^2k)$  and space  $O(|Q|^2)$ , as it involves  $|Q|^2$  steps and, in each step, we run the DFA  $T$  on an input word of length  $k$ . Then, algorithm ASP1( $T, Q, X$ ) works as described in the previous paragraph, but now the test  $\delta^\otimes([v], x) \neq [\text{sink}]$  is reduced to whether  $\text{BQ}[\bar{v}, \bar{x}]$  is true. Hence, assuming that the array BQ is available, the algorithm runs in time  $O(|Q||X|)$ .

### 5.2 Algorithm ASP2( $T, Q, Z, Y$ ) for (SP2)

Here we assume that  $Q_Z^\succ = Y$ , and we compute  $X$  by initializing it to  $Z$ , and then by repeatedly adding into  $X$  a new element from  $V = Q - X$ , provided that condition  $Q_X^\succ = Y$  remains true. In particular, the algorithm is as follows.

```

ASP2(T, Q, Z, Y)
    X = Z; V = Q - X;
    while (V ≠ ∅)
    
```



```

do
  Pick  $v \in V$ ;
  Use ASP1( $T, Q, X \cup \{v\}$ ) to compute  $Y' = Q_{X \cup \{v\}}^>$ ;
  if ( $Y' = Y$ )  $X = X \cup \{v\}$ ;
   $V = V - \{v\}$ ;
return  $X$ ;

```

**Lemma 8.** *The above algorithm computes in time  $O(|Q|^2(|Q| - |Z|))$  a subset  $X$  of  $Q$  such that  $Z \subseteq X$  and  $X$  is maximal with “ $X \subseteq Q$  and  $Q_X^> = Y$ ”.*

In the subsection below on Problem (P1), we give an example of executing Algorithm ASP2 based on input from Fig. 2.

### 5.3 Algorithm ASP3( $T, Q, X, Y$ ) for (SP3)

We assume that  $T = \text{Trie}(S)^\otimes$  is given, as well as, the sets  $Q, X, Y$ . The required DFA  $T'$  accepting  $S^\otimes \cap X\Sigma^* \cap \Sigma^*Y$  can be constructed as follows by modifying  $T$  in *linear time* in terms of the sizes of the given structures.

- The states of  $T'$  are [sink], all states in  $[Q]$ , and all  $[z]$  with  $z$  a prefix of  $X$ .
- The start state is  $[\lambda]$ , and the set of final states is  $[Y]$ .
- The transitions of  $T'$  are all the transitions of  $T$  involving only the above states.

It is not difficult to see that, indeed, the automaton  $T'$  accepts exactly those words in  $S^\otimes$  that end with a suffix in  $Y$  and begin with a prefix in  $X$ , as required.

### 5.4 Algorithm for Problem (P1)

We present now the algorithm for (P1), our first original main problem.

```

A1(S)
  Compute  $T = \text{Trie}(S)^\otimes$ ;
  Compute the strongly connected components of  $T$ ;
  Pick a nontrivial component  $[Q]$  – exit if none exists;
  Compute the Boolean array BQ;
  Compute two nonempty subsets  $Z, Y$  of  $Q$  such that  $Q_Z^> = Y$ ;
  Use ASP2( $T, Q, Z, Y$ ) to compute a maximal  $X$  with  $Q_X^> = Y$ ;
  Use ASP3( $T, Q, X, Y$ ) to compute and return the DFA for  $S^\otimes \cap X\Sigma^* \cap \Sigma^*Y$ ;

```

Step 2 can be computed in linear time in terms of the size of  $\text{Trie}(S)^\otimes$  – see [6]. Steps 3 and 5 are nondeterministic and allow for various possibilities. We show how to do Step 5 in time  $O(|Q|)$ . We pick any  $z \in Q$  and let  $Z = \{z\}$ . We need to show that  $Y = Q_Z^>$  is not empty. By Lemma 5, there is some state  $[u] \in [Q]$  with a path of length  $k$  to  $[z]$ . As  $[z] \neq [\text{sink}]$ , the label of that path must be equal to  $z$ . Hence,  $u \in Q_Z^>$ . By the results in the previous sections, it is easy to see that the above algorithm operates correctly as described in the following theorem. Also, as the set  $Z$  is of cardinality 1, Step 6 of the algorithm runs in time  $O(|Q|^3)$ .

**Theorem 3.** *Algorithm A1(S) computes, for any given subword constraint S of some length k, a DFA accepting a nonempty language D whose words are of length at least k, and D is maximal with  $D^* \subseteq S^\otimes$ ; or the algorithm reports that no such D exists. The algorithm runs in time  $O(|Q|^3 + |Q|^2k)$  and space  $O(|Q|^2k)$ , where  $[Q]$  is any nontrivial SCC of  $\text{Trie}(S)^\otimes$  – if such exists.*

*Example 3.* Going back to Fig. 2, for  $Q = \{0111, 1111, 1110, 1101, 1011\}$ , Step 5 of Algorithm A1(S) can be performed by choosing, for instance,  $Z = \{1011\}$  and computing  $Y = Q_{\{1011\}}^\triangleright = \{1011, 0111, 1111\}$ . Then in Step 6, the word 0111 will not be added to Z, as  $\delta^\otimes([1011], 0111) = [\text{sink}]$ . On the other hand, the words 1101, 1110, 1111 will be added to Z to obtain  $X = \{1011, 1101, 1110, 1111\}$  with  $Q_X^\triangleright = Y$ . Thus, the language

$$D = S^\otimes \cap \{1011, 1101, 1110, 1111\} \Sigma^* \cap \Sigma^* \{1011, 0111, 1111\}$$

is maximal with  $D^* \subseteq S^\otimes$ .

### 5.5 Algorithm for Problem (P2)

The desired algorithm is very similar to the one used for Problem (P1).

A2(S)

- Compute  $T = \text{Trie}(S)^\otimes$ ;
- Compute the strongly connected components of T;
- Pick a component  $[Q]$  containing a fork state – exit if none exists;
- Compute the Boolean array BQ;
- Compute subsets Z, Y of Q such that  $|Z| \geq 2$  and  $Q_Z^\triangleright = Y$ ;
- Use ASP2(T, Q, Z, Y) to compute a maximal X with  $Q_X^\triangleright = Y$ ;
- Use ASP3(T, Q, X, Y) to compute and return the DFA for  $S^\otimes \cap X \Sigma^* \cap \Sigma^* Y$ ;

For the correctness of the algorithm, we first note that there is a nontrivial language D with  $D^* \subseteq S^\otimes$  if and only if there is a SCC of  $\text{Trie}(S)^\otimes$  containing a fork state – the ‘only if’ part follows from Theorem 2 and the ‘if’ part from Lemmata 3 and 6. Thus, if there is no fork state in some SCC, Step 3 correctly decides to terminate the algorithm. On the other hand, if a fork state is found, then, according to Lemma 6, we can define effectively a two-element subset Z of Q such that  $\langle Q, Z \rangle$  is nontrivial. Then, in Step 6, the algorithm attempts to add elements to Z in order to obtain a subset X of Q that is maximal with  $Q_X^\triangleright = Y$ . As before, the resulting set  $\langle Q, X \rangle = S^\otimes \cap X \Sigma^* \cap \Sigma^* Y$  is maximal with  $\langle Q, X \rangle^* \subseteq S^\otimes$  and, of course, the set is also nontrivial as it contains the nontrivial set  $\langle Q, Z \rangle$ . The time complexity of the above algorithm is the same as that of A1(S).

**Theorem 4.** *Algorithm A2(S) computes, for any given subword constraint S of some length k, a DFA accepting a nontrivial language D whose words are of length at least k, and D is maximal with  $D^* \subseteq S^\otimes$ ; or the algorithm reports that no such D exists. The algorithm runs in time  $O(|Q|^3 + |Q|^2k)$  and space  $O(|Q|^2k)$ , where  $[Q]$  is any SCC of  $\text{Trie}(S)^\otimes$  containing a fork state – if such exists.*

*Example 4.* Again in Fig. 2, for  $Q = \{0111, 1111, 1110, 1101, 1011\}$ , we see that  $[0111]$  is a fork state with transitions going to states  $[1110], [1111]$ . Let  $Z = \{1110, 1111\}$ . Step 5 will compute  $Y = Q_Z^> = Q$  and, then, Step 6 will find that no other words will be added to  $Z$ , that is,  $X = Z$ . Thus, the language

$$D = S^{\otimes} \cap \{1110, 1111\} \Sigma^* \cap \Sigma^* \{0111, 1111, 1110, 1101, 1011\}$$

is nontrivial and maximal with  $D^* \subseteq S^{\otimes}$ .

## 6 Concluding Remarks

We have considered the problem of characterizing nontrivial languages  $D$  that are maximal with the property  $D^* \subseteq S^{\otimes}$ . Our characterization is structural and nontrivial, and leads to algorithmically polynomial solutions. The recent work of [12] solves the more general problem of computing all maximal solutions of  $D^* \subseteq R$ , for any given regular language  $R$  using a brute force method of exponential time complexity. Is it possible to dig deeper and combine the two approaches with the aim of computing efficiently the more general problem? Is it possible to compute efficiently nontrivial  $D$ 's that are maximal with the conjunctive property " $D^* \subseteq S^{\otimes}$  and  $D$  is a code (or a prefix code)"? Can we find the right tools for choosing "good", according to some information theory criteria, maximal  $D$ 's among the potentially many ones that exist in  $S^{\otimes}$ ?

## References

1. Campeanu, C., Konstantinidis, S.: State complexity of the subword closure operation with applications to DNA coding. *International Journal of Foundations of Computer Science* 19(5), 1099–1112 (2008)
2. Chen, J., Reif, J. (eds.): *Preproceedings of DNA9, June 2003*. Madison, Wisconsin (2003)
3. Crochemore, M., Hancart, C.: Automata for matching patterns. In: [14], pp. 399–462
4. Cui, B.: Encoding methods for DNA languages defined via the subword closure operation. MSc Thesis, Dept. Math. and Computing Science, Saint Mary's University, Canada (2007)
5. Cui, B., Konstantinidis, S.: DNA coding using the subword closure operation. In: [8], pp. 284–289.
6. Dasgupta, S., Papadimitriou, C.H., Vazirani, U.V.: *Algorithms*. McGraw-Hill, New York (2006)
7. Ferretti, C., Mauri, G., Zandron, C. (eds.): *DNA Computing, 10th International Workshop on DNA Computing, DNA 10, Milan, Italy, Revised Selected Papers*. In: Ferretti, C., Mauri, G., Zandron, C. (eds.) *DNA 2004*. LNCS, vol. 3384. Springer, Heidelberg (2005)
8. Demaine, E.D., Demaine, M.L., Fekete, S.P., Ishaque, M., Rafalin, E., Schweller, R.T., Souvaine, D.L.: Staged self-assembly: Nanomanufacture of arbitrary shapes with  $O(1)$  glues. In: Garzon, M.H., Yan, H. (eds.) *DNA 2007*. LNCS, vol. 4848, pp. 1–14. Springer, Heidelberg (2008)
9. Havel, I.M., Koubek, V. (eds.): *MFCS 1992*. LNCS, vol. 629. Springer, Heidelberg (1992)
10. Jonoska, N., Mahalingam, K.: Languages of DNA based code words. In: [2], pp. 58–68

11. Kari, L., Konstantinidis, S., Sosík, P.: Bond-free languages: formalizations, maximality and construction methods. In: [7], pp. 169–181
12. Kari, L., Seki, S.: Schema for parallel insertion and deletion. In: Gao, Y., Lu, H., Seki, S., Yu, S. (eds.) DLT 2010. LNCS, vol. 6224, pp. 267–278. Springer, Heidelberg (2010)
13. Mahalingam, K.: Involution codes: with application to DNA strand design. PhD Thesis, Department of Mathematics, University of South Florida, Florida, USA (2004)
14. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages, vol. I. Springer, Berlin (1997)
15. Sakarovitch, J.: Elements of Automata Theory. Cambridge University Press, Berlin (2009)
16. Yu, S.: Regular Languages. In: [14], pp. 41–110.

# Modelling, Simulating and Verifying Turing-Powerful Strand Displacement Systems

Matthew R. Lakin and Andrew Phillips

Microsoft Research, Cambridge, CB3 0FB, UK  
aphillip@microsoft.com

**Abstract.** We demonstrate how the DSD programming language can be used to design a DNA stack machine and to analyse its behaviour. Stack machines are of interest because they can efficiently simulate a Turing machine. We extend the semantics of the DSD language to support operations on DNA polymers and use our stack machine design to implement a non-trivial example: a ripple carry adder which can sum two binary numbers of arbitrary size. We use model checking to verify that the ripple carry adder executes correctly on a range of inputs. This provides the first opportunity to assess the correctness and kinetic properties of DNA strand displacement systems performing Turing-powerful symbolic computation.

## 1 Introduction

Biomolecular computation devices can interface directly with living tissue [1], opening up exciting new possibilities for autonomous medical intervention at the cellular level. The programmable nature of DNA makes it ideally suited as a material to implement such biomolecular computers. As techniques for DNA synthesis and manipulation continue to improve, we can look towards using DNA to implement more sophisticated computational functions.

Classical work on computability theory has produced a number of equivalent universal computational models, such as Turing machines [2] and stack machines. Both of these paradigms are based on symbolic computation, where computation proceeds via the manipulation of abstract mathematical symbols which denote data values. These paradigms have the virtues of simplicity and compactness, as simple data structures are modified in-place. Nucleic acids are excellent materials for implementing symbolic computation, because distinct symbols can be straightforwardly represented as distinct, non-interfering nucleotide sequences, and data structures can be directly realized in the physical structure of the DNA species.

In this paper we study the design and analysis of biomolecular implementations of universal symbolic computation. Our chosen framework for molecular computation is DNA strand displacement [3], which is an established technique for the principled design of DNA computing systems. Our starting point is the work of Qian et al. [4], who proposed a design of a stack machine using DNA strand displacement. A stack machine consists of finitely many stacks (first-in,

first-out memory storage) and a finite state machine which can add symbols to (push), and remove symbols from (pop), the top of these stacks. In [4] the stack data structures have a direct physical representation as DNA polymers which can interact at one end only. This design is a simple and elegant translation of a universal scheme for symbolic computation into DNA, which can be used to efficiently simulate a Turing machine [2].

We tackle the formal design and analysis of universal DNA computers by encoding them in the DSD programming language [5]. This is a domain-specific language with a well-defined operational semantics that reflects the key assumptions of strand displacement systems. DSD has previously been used to model a range of strand displacement devices, including logic gates and chemical reaction networks. However, previous versions of DSD did not support the formation of extensible polymers, which are required to encode stack data structures in DNA. Furthermore, the DSD simulation algorithm required all models to be compiled to a fixed set of reactions and was therefore unable to simulate Turing-powerful computation, which can generate potentially unbounded numbers of reactions. Hence we extend the DSD semantics and simulation algorithm to support the formation of linear heteropolymers.

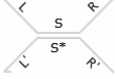
This paper is structured as follows. Section 2 presents an extension of the DSD language syntax and semantics [5] to model the formation of linear DNA heteropolymers, while Section 3 presents a stochastic simulation algorithm, based on [6], for DNA strand displacement systems involving polymers. Section 4 presents an encoding of a stack machine design in the DSD language which is optimised for mechanical verification. Finally, Section 5 presents an implementation of a classic circuit from digital electronics, a ripple carry adder, which computes the sum of two binary numbers of arbitrary size, including results from stochastic simulations and model-checking which provide evidence that the DNA implementation of the adder is correct. To our knowledge, this is the largest DNA strand displacement system to be formally verified.

## 2 Polymers in DSD

The DSD language was introduced in [5] as a means of formalising the designs of DNA strand displacement systems. Here we recap the basics and extend the semantics to allow polymerisation reactions between complexes.

The syntax of the DSD language is defined in terms of *domains*  $M$  and *domain sequences*  $S$ ,  $L$ ,  $R$ . A domain  $M$  represents a nucleotide sequence with explicit information about the orientation of its 3' and 5' ends. We assume that distinct domains are mapped to distinct, non-interfering nucleotide sequences using established techniques [7]. A domain can be a *long domain*  $N$  or a *short domain*  $N^\sim$  (shown in black in images). We assume that toeholds are sufficiently short to hybridize reversibly (4–10nt) whereas long domains are sufficiently long to hybridize irreversibly (>20nt). A domain sequence  $S$  is a concatenation of finitely many domains with the same orientation, whereas domain sequences  $L$  and  $R$  can potentially be empty. The *complement*  $S^*$  of a domain sequence  $S$  is the domain sequence that hybridizes with  $S$  via Watson-Crick complementarity.

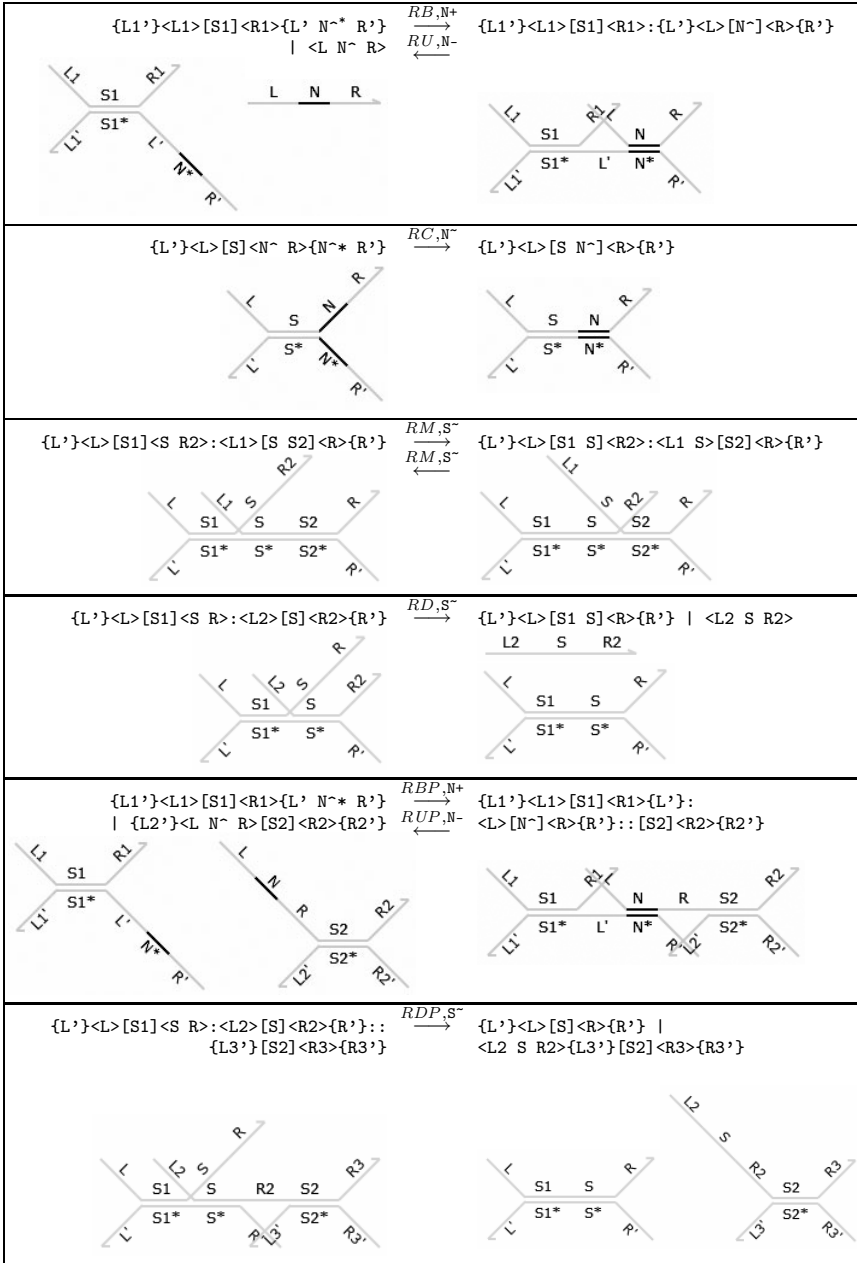
**Table 1.** Graphical and textual syntax of the DSD programming language

Syntax	Description	Syntax	Description
$\{\underline{S}\}$	Lower strand with sequence $S$	$\{L'\}\langle L\rangle[S]\langle R\rangle\{R'\}$	Double stranded complex $[S]$ with overhanging single strands $\langle L\rangle$ , $\langle R\rangle$ and $\{L'\}$ , $\{R'\}$
$\langle \underline{S} \rangle$	Upper strand with sequence $S$		
$C1:C2$	Complexes joined by lower strand	$C1::C2$	Complexes joined by upper strand

Domain sequences are used to construct DNA species, as shown in Table 1. A species can either be a single *strand*  $A$  or a *complex*  $C$ . A strand can either be an *upper* strand  $\langle S \rangle$  (drawn with the 3' end towards the right) or a *lower* strand  $\{S\}$  (drawn with the 3' end towards the left). We assume that species are equal up to rotation symmetry, so every upper strand has a corresponding lower strand, and vice versa. Complexes are formed by joining one or more *segments* of the form  $\{L'\}\langle L\rangle[S]\langle R\rangle\{R'\}$ , which consists of a double-stranded region  $[S]$  with four overhanging strands. This represents an upper strand  $\langle L S R \rangle$  bound to a lower strand  $\{L' S^* R'\}$  by hybridization between  $S$  and  $S^*$ . For compactness, only the upper sequence of the double-stranded region is written explicitly, and we omit empty overhanging strands. Complexes can be formed by concatenating segments either along the lower strand, written  $C1:C2$ , or along the upper strand, written  $C1::C2$ .

Systems  $D$  typically involve many species in parallel, written  $D_1 \mid \dots \mid D_n$ . We abbreviate  $K$  parallel copies of the same system  $D$  as  $K*D$ . The language also includes features for expressing the logical structure of the system: a domain  $N$  can be restricted to the system  $D$ , written **new**  $N D$ , which represents the assumption that  $N$  and  $N^*$  do not appear outside of  $D$ . The language also supports module definitions of the form  $X(\tilde{m})=D$ , where  $\tilde{m}$  is a list of module parameters and  $X(\tilde{n})$  is an instance of the module  $X$  with the parameters  $\tilde{m}$  replaced by values  $\tilde{n}$ . We assume a fixed collection of non-recursive module definitions. A key assumption of the DSD language is that species only interact via complementary toeholds: we enforce this by requiring that no long domain and its complement are simultaneously exposed. Finally, we note that the syntax of the DSD language is constrained so that overhanging single strands are the only secondary structure which a complex may possess, which rules out branching structures.

Figure 1 presents elementary reduction rules for the DSD language which formalise basic strand displacement reactions. Rules (RB) and (RU) define the *binding* of a strand to a complex via a complementary toehold, together with the corresponding *unbinding* reaction since we assume that toeholds hybridize reversibly. The rates of these reactions are determined from the toehold  $N$ . Rule (RC) accounts for the case when an overhanging toehold in the lower strand is *covered* by the complementary toehold in the upper strand: this is irreversible as the resulting long double-stranded segment is thermodynamically stable. Rules



**Fig. 1.** Elementary reduction rules of the DSD language with polymers. We let  $S^-$  denote the migration rate of a domain sequence  $S$ , and we let  $N^+$  and  $N^-$  denote the binding and unbinding rates, respectively, of a toehold  $N^*$ . We assume that  $\text{fst}(R2) \neq \text{fst}(S2)$  for rule (RM). This ensures that branch migration is maximal along a given sequence and that rules (RM) and (RD) are mutually exclusive.



(RM) and (RD) define *branch migration* and *strand displacement* reactions, respectively. In each of these, the overhanging junction in the upper strand performs a random walk which, in the case of rule (RD), completely displaces a strand from the complex. Note that branch migration is a reversible process whereas strand displacement is irreversible.

The final two rows in Figure 1 present additional reduction rules which do not feature in previous published semantics for the DSD language [5]. These rules permit complexes to interact with each other to form larger complexes which we refer to as polymers. Rule (RBP) allows two complexes to bind on a shared toehold to form a longer complex, and rule (RUP) allows the larger complex to break apart when the toehold unbinds. Rule (RDP) extends the strand displacement rule (RD) to the case where the displaced strand was previously holding two complexes together. Note that the reduction rules ensure that the only toeholds which may interact are located in the main trunk of the complex as opposed to in the overhangs: this prevents the formation of branching structures while permitting the growth of linear heteropolymers.

The rules presented in Figure 1 define the basic forms of reduction in the extended DSD language. However, these reactions may take place within larger contexts, so to complete the language semantics we require some additional contextual rules. These include adding segments on either side of the reacting segment, mirroring the species horizontally and vertically, and rotating them. We omit the contextual rules here for reasons of space. In the case of rule (RBP), we note that the overhangs containing the complementary toeholds must appear at the very ends of the complexes: in other words, polymers can only interact end-to-end. This can be formalised by a careful choice of contextual rules which only allow additional structure at one end of interacting complexes. This restriction is necessary to prevent branching structures from arising dynamically.

### 3 Stochastic Simulation of Polymerising Systems

The standard Gillespie algorithm for exact stochastic simulation [8] requires that the entire chemical reaction network (CRN) of all possible reactions between reachable chemical species must be known before the simulation begins. However, in the case of DNA strand displacement systems with polymers we cannot necessarily pre-compute the CRN because it may be infinite, as we could (in principle) keep adding monomers to produce an ever-increasing polymer chain.

We avoid this problem by using the just-in-time simulation algorithm from [6]. This is an extension of the Gillespie algorithm in which compilation of species interactions is interleaved with simulation steps. The just-in-time simulation algorithm can be summarised as follows:

1. Compute the CRN of all possible initial reactions between the initial species only, *without* recursively computing reactions involving the products of those initial reactions.
2. Compute reaction propensities according to the Gillespie algorithm, and randomly select the next reaction with probability proportional to its propensity.

3. Execute the next reaction by modifying the species populations and incrementing the simulation time according to the Gillespie algorithm.
4. If executing the reaction produced any new species which have not yet been seen in the system then compute any interactions between the new species and the existing species in the system, and expand the CRN accordingly.
5. Repeat from step 2.

Thus we dynamically update the set of possible reactions as the simulation proceeds, rather than computing all possible reactions up front. Hence we compute only the needed subgraph of the CRN. This can offer significant speedups when the CRN is very large, and is the only feasible approach when the CRN is infinite, as in the case of most polymerising systems. The stochastic simulation is exact since all probabilities are computed exactly at each step.

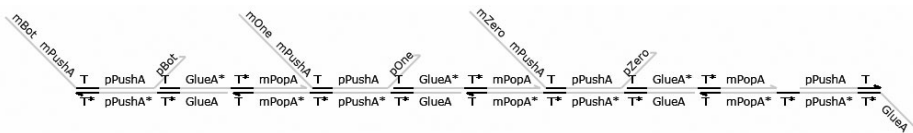
## 4 Modelling Stack Machines in DSD

In this section we present a novel stack design which is a variant of the stack encoding from [4]. Our design was formalised, visualised and analysed using the Visual DSD tool<sup>1</sup>. Our primary goal in designing a new stack implementation is to produce an encoding which is amenable to automated verification. Thus we aim to eliminate speculative stack manipulation reactions and irreversible steps in reaction gates which could occur at any time after the outputs are produced.

### 4.1 A Variant Stack Encoding

The stack design from [4] has the property that fuel monomers specific to the various symbols that might be pushed onto the stack are continually interacting with the stack, in the hope that the symbol strand itself may arrive to complete the reaction, as in Figure 3 of [4]. Furthermore, that Figure shows that the fuel strands which can deconstruct the stack are also continually interacting with the stack, in the hope that other species may arrive to complete the reaction. This means that there are always a large number of possible stack-based interactions, and consequently the graph of possible states for these systems is very large indeed, making it infeasible to perform analyses such as model checking on the resulting CTMC.

In order to efficiently simulate a Turing machine, more than one stack is needed for data storage. Thus we must assign a unique *type* to each stack so that they can be correctly addressed. In our stack encoding, the stack  $[[::1::0]$  of type *A*, is represented by the following DNA complex. Note that we write the top of the stack on the right-hand side in our textual notation, to match the visualisations.



<sup>1</sup> Visual DSD is available online at <http://research.microsoft.com/dna>.

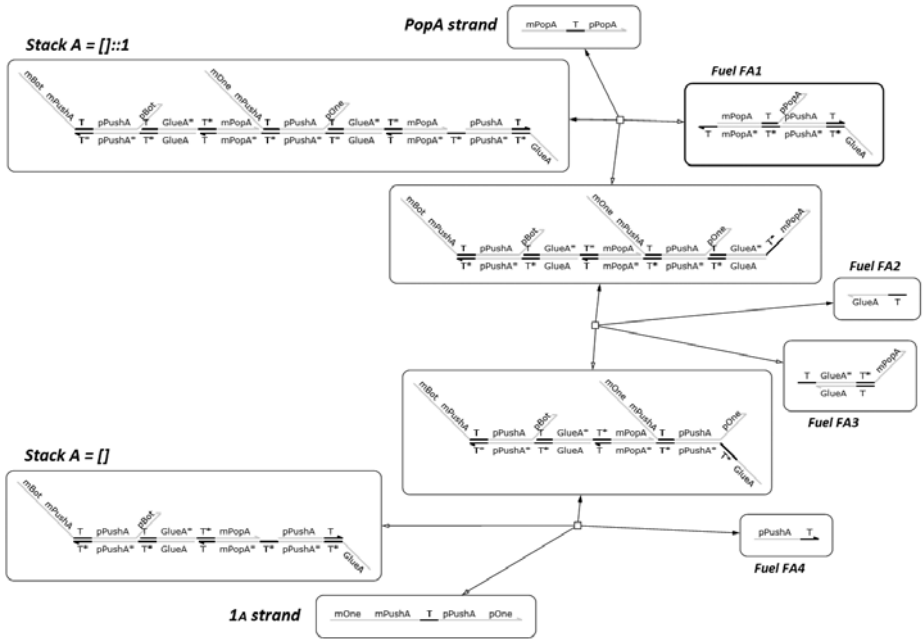
In our encoding, a symbol 1 on stack  $A$  is represented by a bound upper strand of the form  $\langle \text{mOne mPushA T}^{\sim} \text{pPushA pOne} \rangle$ , which we refer to as the *push strand*  $1_A$ . In this paper we use three kinds of symbol: a special *bottom* symbol  $\perp$  which signals an attempt to pop from an empty stack and two symbols corresponding to 1 and 0 respectively. Symbols are represented using a history-free scheme similar to that used in [4], except that we separate the nucleotide sequences on either side of the toehold into two long domains: one specific to the stack type ( $A$  here) and one specific to the symbol in question. We restrict ourselves to ASCII syntax, writing  $\text{mX}$  and  $\text{pX}$  for the negative (towards the 5' end) and positive (towards the 3' end) sequences, which were referred to as  $^-X$  and  $^+X$  respectively in [4].

Each stack complex has a single exposed  $\text{T}^{\sim}$  toehold, which serves as the initiation site for both *push* and *pop* reactions. The reaction to *pop* a symbol from stack  $A$  is initiated by the *pop strand*  $\text{PopA} = \langle \text{mPopA T}^{\sim} \text{pPopA} \rangle$  which begins the clockwise sequence of reversible reactions shown in Figure 2. The fuel species FA1–4 are assumed to be present in abundance. Overall, these reversible reactions interconvert between the stack  $A = [::1$  and the  $\text{PopA}$  strand, and the stack  $A = []$  and the  $1_A$  strand. The *push* reaction is initiated by a push strand and is obtained as the reverse of the above reaction scheme, reading anti-clockwise in Figure 2. When attempting to pop from an empty stack the reactions proceed as in Figure 2, except that the resulting complex is not a valid stack structure. The *bottom strand*  $\perp_A = \langle \text{mBot mPushA T}^{\sim} \text{pPushA pBot} \rangle$  serves as an error indicator, signalling that an attempt has been made to pop from an empty stack.

Our stack design allows us to initiate pushing or popping by the interaction of a single strand with a stable stack complex, without speculative binding and unbinding reactions as in [4]. Furthermore, we can use a smaller set of backbone monomers for each stack type: for a given stack type  $A$  we only require the four fuel species FA1–4 from Figure 2 because any symbol can be joined to the main backbone of the stack by the common  $\text{pPushA}$  domain. Hence the number of domains required scales with the sum of the number of stacks and the number of symbols, whereas in the encoding of [4] it scales with the product (because there the separate nucleotide sequences denoting the stack and the symbol parts of the  $\langle \text{mOne mPushA T}^{\sim} \text{pPushA pOne} \rangle$  strand are merged so the strand has the form  $\langle \text{mOneA T}^{\sim} \text{pOneA} \rangle$ ). In our design it is crucial that the pop strand employs the history-free encoding from [4], so it can initiate a leftward displacement reaction to break apart the polymer structure.

## 4.2 Implementing a Stack Machine in DNA

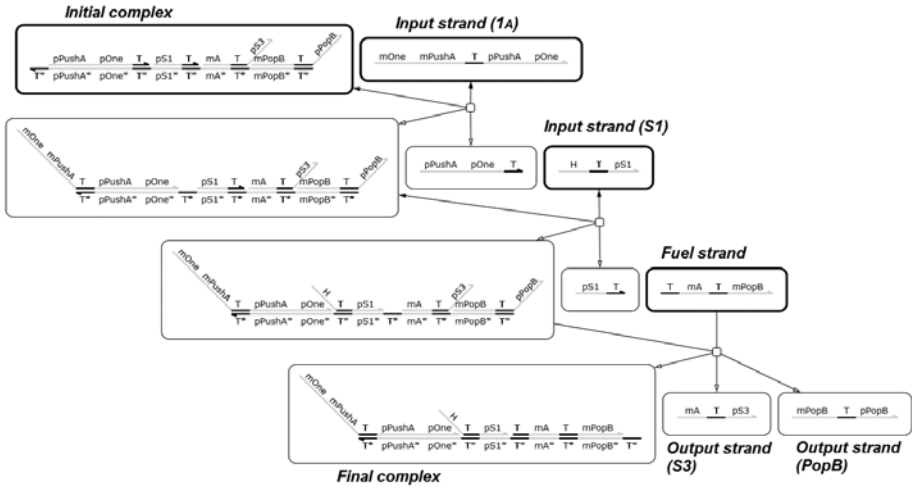
A stack machine consists of finitely many stacks along with a finite state control program. Thus, a configuration of a stack machine consists of the current state and the current contents of the stacks. As discussed above, the symbols in a given stack are encoded in the nucleotide sequences of the overhanging single strands attached to the polymer backbone of the corresponding stack complex. We encode the current state of the machine by a single complex of the form



**Fig. 2.** Example CRN for reversible stack manipulation reactions: pushing and popping a non-empty stack

$S1 = \langle H \ T \ pS1 \rangle$ , where we refer to  $H$  as the history domain and where  $pS1$  is a domain which informs us that the machine is currently in state 1. The history domain is irrelevant when determining the current state of the system, and we will see below why we allow state strands to have an arbitrary history domain. We require that only one state strand is present in solution at any one time, so there can be no confusion over the current state of the machine.

Stack manipulation operations are implemented as described in Section 4.1, and we encode state transitions using chemical reaction gates. These accept as input the current state strand and the output strand from the stack manipulation reaction occurring in that state and produce as output the state strand and stack manipulation initiator strand corresponding to the next state according to the stack machine program. Figure 3 presents the CRN for a reaction gate implementing the reaction  $1_A + S1 \longrightarrow S3 + PopB$ , which assumes that the symbol 1 has just been read from stack  $A$  in state 1, and the transition is to state 3 where we must pop from stack  $B$ . In the CRN, the bold nodes denote the species initially present. This gate accepts the input strands  $1_A = \langle mOne \ mPushA \ T \ pPushA \ pOne \rangle$  and  $S1 = \langle H \ T \ pS1 \rangle$  (where  $H$  is an arbitrary history domain) and produces the output strands  $S3 = \langle mA \ T \ pS3 \rangle$  and  $PopB = \langle mPopB \ T \ pPopB \rangle$ . Here, the domain  $mA$  is a private history domain which is unique to this particular reaction gate. This allows us to use a long fuel strand with multiple toeholds to eject both of the outputs and render the



**Fig. 3.** Example CRN for an irreversible stack machine transition. Nodes with a bold outline indicate species required to be present initially.

complex unreactive in a single step. This helps to restrict the number of states in the CTMC because the chemical reactions corresponding to different steps of the stack machine computation are separated by these irreversible displacement reactions. We are left with a final complex which we consider to be unreactive because there is no other species in the system which can displace the entirety of the long fuel strand. We do not add an extra toehold to the fuel strand to completely seal off the complex because this can lead to unwanted interference caused by fuel strands reacting with the stack monomers.

A stack machine terminates when it enters an *accepting* or *rejecting* state, which can have no outgoing transitions. State transitions which enter one of these states are implemented using a reaction gate similar to that from Figure 3, except that in this case we *can* add an extra toehold to the fuel strand in order to completely seal off the final complex without causing unwanted interference. Reaction gates which implement transitions into an accepting or rejecting state do not produce a strand to initiate another stack operation and this, together with the fact that the fuel strand completely seals off the complex, means that all chemical reactions in the system cease. Hence the CTMC has a well-defined terminal state from which no reactions are possible. This makes it more convenient to ask questions about the final state of the machine.

### 5 DSD Stack Machine Example: Ripple Carry Adder

As a non-trivial proof of concept we implemented a binary adder in DSD using the stack machine encoding described above. Figure 4 presents the stack machine program for a ripple carry adder which iteratively sums the corresponding bits

from two binary numbers while maintaining a carry bit. The binary numbers are stored in stacks by using different symbols to denote 0 and 1. States which involve popping from a stack have three outgoing transitions (depending on whether one, zero or bottom was popped) and states which involve pushing onto a stack have just one outgoing transition. For the sake of clarity, the state graph in Figure 4 omits a rejecting state along with the transitions into this state: the missing transitions are from state 2 when 0 is popped from stack  $B$ , from states 3 and 4 when  $B = []$  and from states 5, 6 and 7 when  $C = []$ . These transitions signal an error when the two inputs are of different lengths or when the carry bit is not present as expected.

The machine reads input from stacks  $A$  and  $B$  (without loss of generality we assume that both inputs comprise the same number of bits) and takes its carry bit from  $C$  (initially zero). For each pair of input bits the stack machine implements a full adder and by iterating the loop we get the effect of a ripple carry adder. When it terminates, the machine has written the sum of the two inputs into  $X$  along with a carry-out bit in  $C$ . Due to the first-in, first-out nature of the stack data structure, the endianness of the output in  $X$  is flipped relative to that of the inputs, though this could be rectified by a subsequent reversing operation if necessary.

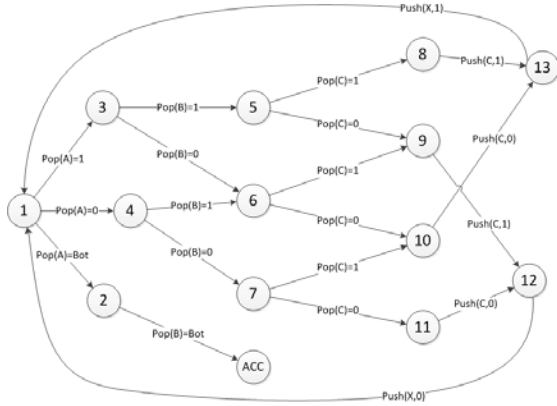
## 5.1 Stochastic Simulation

Figure 5 presents an example<sup>2</sup> of a stochastic simulation for 1-bit addition with inputs  $A = []::1$ ,  $B = []::0$  and  $C = []::0$ . This plot was obtained using the simulation algorithm described in Section 3. It shows which of the state strands has population 1 at a given time during the run, which allows us to trace the execution of the machine. Comparing the sequence of states from this timeline with the state diagram from Figure 4 shows us that the machine did in fact go through the expected sequence of states. Furthermore, the contents of the output stacks at the end of this simulation run were  $X = []::1$  and  $C = []::0$ , which agrees with the truth table from Figure 4. Thus we have some preliminary evidence that our stack machine program is working correctly.

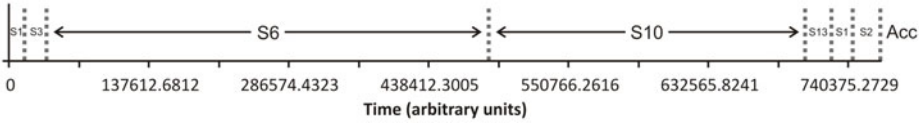
The simulation plot from Figure 5 also gives us some information regarding the kinetic behaviour of the stack machine implementation. In particular, we observe that the machine spends far longer in states 6 and 10 than in any of the other states. These bad kinetics are caused by the excess of reaction complexes relative to the single stack complex. If a strand could bind either to a stack or to a reaction complex, it will be far more likely to bind to the reaction complex as they are present in excess. We can attenuate this effect to an extent in our simulations by reducing the population of fuels. However, we must strike a balance between providing enough fuel to finish the computation and maintaining reasonable kinetics. Furthermore, in general computations may be arbitrarily long and we may not know the optimal amount of fuel in advance. This is not an

<sup>2</sup> DSD and PRISM source code for the models discussed in Section 5 are available online at <http://research.microsoft.com/dna/dna17.zip>.

$C_{in}$	$A$	$B$	$X$	$C_{out}$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



**Fig. 4.** (Left) Truth table for a 1-bit full adder, which takes two bits and a carry bit as input and produces an output bit and a carry output bit. (Right) State diagram for a stack machine implementation of a ripple carry adder, where state 1 is the initial state.



**Fig. 5.** Example stochastic simulation plot showing the populations of state strands during an execution of the ripple carry adder stack machine. The populations switch between zero and one as the stack machine moves through the sequence states defined by its program.

artefact of our stack machine encoding—the issue also exists with the original design proposed in [4]. However, in that paper there was no stochastic simulation available to observe the kinetic behaviour of the stack machine.

### 5.2 Model Checking

We used the PRISM model checker [9] to verify that the ripple carry adder, given particular inputs, satisfies certain properties expressed as temporal logic formulae. To demonstrate that the stack machine works correctly for given inputs, we used PRISM to check that the following properties hold of the CTMC of the system. We give informal descriptions as well as example PRISM queries:

1. the system always goes through the correct sequence of state transitions and eventually reaches a terminal state which contains the expected output species (P=? [F(state\_is\_X & F( ... & F(state\_is\_Y & ‘deadlock’ & outputs\_correct) ... )]);

Input A			Input B			Output X		Output C	Result
MSB	LSB	Value	MSB	LSB	Value	LSB	MSB	Value	Value
0	0	0	0	0	0	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>
0	0	0	0	1	1	<b>1</b>	<b>0</b>	<b>0</b>	<b>1</b>
0	0	0	1	0	2	<b>0</b>	<b>1</b>	<b>0</b>	<b>2</b>
0	0	0	1	1	3	<b>1</b>	<b>1</b>	<b>0</b>	<b>3</b>
0	1	1	0	0	0	<b>1</b>	<b>0</b>	<b>0</b>	<b>1</b>
0	1	1	0	1	1	<b>0</b>	<b>1</b>	<b>0</b>	<b>2</b>
0	1	1	1	0	2	<b>1</b>	<b>1</b>	<b>0</b>	<b>3</b>
0	1	1	1	1	3	<b>0</b>	<b>0</b>	<b>1</b>	<b>4</b>
1	0	2	0	0	0	<b>0</b>	<b>1</b>	<b>0</b>	<b>2</b>
1	0	2	0	1	1	<b>1</b>	<b>1</b>	<b>0</b>	<b>3</b>
1	0	2	1	0	2	<b>0</b>	<b>0</b>	<b>1</b>	<b>4</b>
1	0	2	1	1	3	<b>1</b>	<b>0</b>	<b>1</b>	<b>5</b>
1	1	3	0	0	0	<b>1</b>	<b>1</b>	<b>0</b>	<b>3</b>
1	1	3	0	1	1	<b>0</b>	<b>0</b>	<b>1</b>	<b>4</b>
1	1	3	1	0	2	<b>1</b>	<b>0</b>	<b>1</b>	<b>5</b>
1	1	3	1	1	3	<b>0</b>	<b>1</b>	<b>1</b>	<b>6</b>

**Fig. 6.** Table of verification results for all possible pairs of 2-bit inputs to the ripple carry adder (with initial carry bit zero). Output values in **boldface** were computed using PRISM and are known to be the final state of the system irrespective of which particular trajectory the system follows.

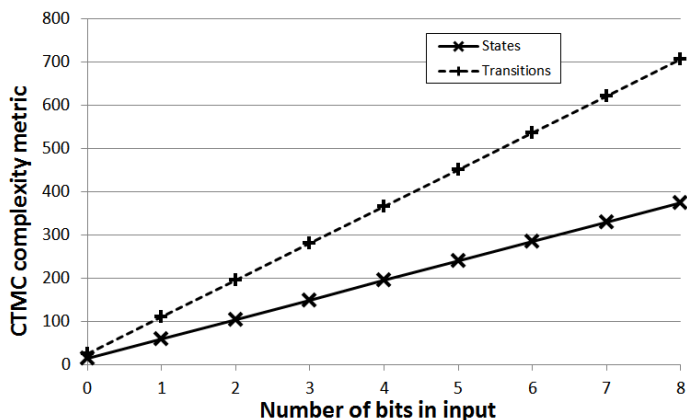
2. there is always precisely one complex for each stack type (`stack_X=1`); and
3. there is always *at most* one state strand (`state_strands<=1`). This is not an equality because the state strand may be bound to a reaction complex.

In the above examples, `outputs_correct` returns true if the state contains the expected output species and `state_is_X` returns true if a state strand corresponding to state  $X$  is present in solution. The “`deadlock`” label identifies a terminal state of the CTMC and `stack_X` and `state_strands` return the populations of all stack complexes corresponding to stack  $X$  and the total population of state strands, respectively. The temporal logic formula  $F\phi$  holds if the system must eventually reach a state satisfying  $\phi$ .

We used PRISM to verify the correctness of all possible pairs of 2-bit inputs (with the initial carry bit set to zero). The results are presented in Figure 6. We were able to show that all four properties listed above hold for all 16 different input pairings, and that we observe the correct output species in the terminal state in all cases. We similarly verified a larger system with two 8-bit inputs, to show that the model checking approach can scale to larger inputs.

Finally, Figure 7 shows how the numbers of states and transitions in the CTMC scale with the initial number of bits in the inputs stacks  $A$  and  $B$ . Thanks to our reaction gate design we see linear increases in numbers of both states and transitions with increasing input size.





**Fig. 7.** CTMC complexity metrics. Each point was calculated for a single pair of inputs of that size: the values of the metrics are identical or very similar for different inputs of the same size.

## 6 Related Work

Theoretical work on the computational power of stochastic chemical reaction networks has shown that chemical systems with polymerisation are Turing-powerful [10,11] but also that *finite* stochastic chemical reaction networks can simulate register machines (and hence Turing machines) with an arbitrarily small probability of error [12,13]. The trick here is to use the *populations* of certain species to denote the numerical values stored in the registers. Jiang et al. [14] have demonstrated how imperative code (which may include arithmetic and while-loops) can be compiled down to stochastic chemical reactions, again using molecular populations to store numerical values. This approach relies on a chemical clock signal to synchronise operations, in order to minimise errors. It is believed that this combination of features is sufficient to make the system Turing-powerful.

Turning to symbolic approaches, Rothmund [15] proposed a design for a universal Turing machine which uses restriction enzymes and ligases to perform operations on a tape encoded as a double-stranded DNA complex. We have already cited the stack machine encoding proposed by Qian et al. [4] as the inspiration for the work reported in this paper.

## 7 Discussion

From an experimental viewpoint, the main issue with the stack machine designs presented in this paper and in [4] is that they call for a single complex to represent each stack. This is problematic for a number of reasons: it is difficult to produce a single complex with a given design in the lab and it introduces numerous points of failure into the system. If one stack becomes corrupted or forms unwanted secondary structure then the whole system fails. Thus it would be desirable

to invent an alternative stack machine design in which there are many copies of each stack complex (and many copies of the state strand) and the updates to the stacks are synchronised, for example using a clock signal such as that proposed in [14]. This would probably require a different scheme for representing stacks, because the reversible stack manipulation primitives used above, and in [4], mean that stack operations could be undone before the synchronisation actually occurs.

We noted in Section 5.1 that increasing the initial populations of fuels, in order to enable long-running computations, can have adverse effects on the simulation kinetics. In the model this can be addressed by using the `constant` keyword of the DSD language to specify that the populations of certain species (such as fuels) should be fixed throughout the simulation. In practice, a more complex experimental setup would be required in which the population of fuels can be replenished, either continually or at regular intervals. In principle, constant replenishment of DNA fuel should allow long-running, or even unbounded, computations (assuming that all computation steps are error-free).

In Section 5.2 we used model checking to provide some formal verification that our stack machine examples work as expected. We were able to demonstrate some scalability by similarly verifying the result of adding a pair of eight-bit inputs. As shown in Figure 7, the size of the CTMC for our stack machine programs varies linearly with the sizes of the inputs. This was a key goal which motivated various design choices, such as the use of private history domains on the single strands which denote the current state of the machine. In general, however, the brute force approach to model checking does not scale to large systems with many different species and large populations. It may be possible to exploit work on modular model checking [16] to avoid this problem.

Another limitation of model checking is that it only verifies properties of the collection of starting species. We were able to verify that all 2-bit inputs are summed correctly, but we cannot derive a proof that the ripple carry adder works correctly for all input sizes. We would probably need to use an interactive theorem prover to prove such results mechanically. This would require formalising the DSD language in said theorem prover, which could be a valuable exercise in itself.

**Acknowledgements.** We thank Dave Parker for help using PRISM, and Erik Winfree and Lulu Qian for useful discussions on the stack machine design from [4].

## References

1. Venkataraman, S., Dirks, R.M., Ueda, C.T., Pierce, N.A.: Selective cell death mediated by small conditional RNAs. *Proc. Natl. Acad. Sci. U S A* 107(39), 16777–16782 (2010)
2. Turing, A.M.: On computable numbers, with an application to the Entscheidungsproblem. *Proc. London Mathematical Society* s2-42(1), 230–265 (1937)
3. Zhang, D.Y., Seelig, G.: Dynamic DNA nanotechnology using strand-displacement reactions. *Nat. Chem.* 3, 103–113 (2011)

4. Qian, L., Soloveichik, D., Winfree, E.: Efficient turing-universal computation with DNA polymers. In: Sakakibara, Y., Mi, Y. (eds.) DNA 16 2010. LNCS, vol. 6518, pp. 123–140. Springer, Heidelberg (2011)
5. Phillips, A., Cardelli, L.: A programming language for composable DNA circuits. *J. R. Soc. Interface* 6(suppl 4), S419–S436 (2009)
6. Paulevé, L., Youssef, S., Lakin, M.R., Phillips, A.: A generic abstract machine for stochastic process calculi. In: Proc. CMSB 2010, pp. 43–54. ACM, New York (2010)
7. Zhang, D.Y.: Towards domain-based sequence design for DNA strand displacement reactions. In: Sakakibara, Y., Mi, Y. (eds.) DNA 16 2010. LNCS, vol. 6518, pp. 162–175. Springer, Heidelberg (2011)
8. Gillespie, D.T.: Exact stochastic simulation of coupled chemical reactions. *J. Phys. Chem.* 115, 1716–1733 (2001)
9. Hinton, A., Kwiatkowska, M., Norman, G., Parker, D.: PRISM: A tool for automatic verification of probabilistic systems. In: Hermanns, H., Palsberg, J. (eds.) TACAS 2006. LNCS, vol. 3920, pp. 441–444. Springer, Heidelberg (2006)
10. Bennett, C.H.: The thermodynamics of computation—a review. *Int. J. Theor. Phys.* 21(12), 905–939 (1982)
11. Cardelli, L., Zavattaro, G.: Turing universality of the biochemical ground form. *Math. Struct. Comp. Sci.* 20(1), 45–73 (2010)
12. Soloveichik, D., Cook, M., Winfree, E., Bruck, J.: Computation with finite stochastic chemical reaction networks. *Nat. Comput.* 7, 615–633 (2008)
13. Cook, M., Soloveichik, D., Winfree, E., Bruck, J.: Programmability of chemical reaction networks. In: Condon, A., Harel, D., Kok, J.N., Salomaa, A., Winfree, E. (eds.) *Algorithmic Bioprocesses*, pp. 543–584. Springer, Heidelberg (2009)
14. Jiang, H., Riedel, M.D., Parhi, K.K.: Synchronous sequential computation with molecular reactions. In: *Design Automation Conference*, San Diego, California, USA, June 5–10 (2011)
15. Rothmund, P.W.K.: A DNA and restriction enzyme implementation of Turing machines. In: Lipton, R.J., Baum, E.B. (eds.) *DNA Based Computers: DIMACS Workshop*, held April 4, pp. 75–120. American Mathematical Society, Providence (1996)
16. Kupferman, O., Vardi, M.Y.: An automata-theoretic approach to modular model checking. *ACM T. Progr. Lang. Sys.* 22(1), 87–128 (2000)

# Synthesizing Small and Reliable Tile Sets for Patterned DNA Self-assembly

Tuomo Lempiäinen, Eugen Czeizler, and Pekka Orponen

Department of Information and Computer Science  
Helsinki Institute for Information Technology HIIT  
Aalto University School of Science  
P.O. Box 15400, FI-00076 Aalto, Finland  
firstname.lastname@aalto.fi

**Abstract.** We consider the problem of finding, for a given 2D pattern of colored tiles, a minimal set of tile types self-assembling to this pattern in the abstract Tile Assembly Model of Winfree (1998). This Patterned self-Assembly Tile set Synthesis (PATS) problem was first introduced by Ma and Lombardi (2008), and subsequently studied by Göös and Orponen (2011), who presented an exhaustive partition-search branch-and-bound algorithm (briefly PS-BB) for it. However, finding the true minimal tile sets is very time consuming, and PS-BB is not well-suited for finding small but not necessarily minimal solutions. In this paper, we modify the basic partition-search framework by using a heuristic to optimize the order in which the algorithm traverses its search space. We find that by running several parallel instances of the modified algorithm PS-H, the search time for small tile sets can be shortened considerably. We also introduce a method for computing the reliability of a tile set, i.e. the probability of its error-free self-assembly to the target tiling, based on Winfree’s analysis of the kinetic Tile Assembly Model (1998). We present data on the reliability of tile sets found by the algorithms and find that also here PS-H constitutes a significant improvement over PS-BB.

## 1 Introduction

Self-assembly of nanostructures templated on synthetic DNA has been proposed by several authors as a potentially ground-breaking technology for the manufacture of next-generation circuits, devices, and materials [4,9,14,16]. Also laboratory techniques for synthesizing the requisite 2D DNA template lattices, many based on Rothemund’s [12] DNA origami tiles, have recently been demonstrated by many groups [6,10].

In order to support the manufacture of aperiodic structures, such as electronic circuit designs, these DNA templates need to be addressable. When the template is constructed as a tiling from a family of DNA origami (or other kinds of) tiles, one can view the base tiles as being “colored” according to their different functionalities, and the completed template implementing a desired color pattern. Now a given target pattern can be assembled from many different families of

base tiles, and it is clearly advantageous to try to minimize the number of tile types needed and/or maximize the probability that they self-assemble to the desired pattern, given some model of tiling errors.

The task of minimizing the number of DNA tile types required to implement a given 2D pattern was identified by Ma and Lombardi [8], who formulated it as a combinatorial optimization problem, the *Patterned self-Assembly Tile set Synthesis* (PATS) problem. Ma and Lombardi proposed two greedy heuristics for solving the task, and subsequently Göös and Orponen [3] presented an exhaustive partition-search branch-and-bound algorithm for it. While the search algorithm presented in [3], which we denote here as PS-BB, is somewhat successful in finding minimal tile sets for small patterns, the size of the search space grows so rapidly that it seems to hit a complexity barrier at approximately pattern sizes of  $7 \times 7$  tiles. In practice one would of course not need to find an absolutely minimal tile set for a given pattern, but any reasonably small solution set would suffice. However, when the algorithm PS-BB fails to find a minimal solution, it does not seem to yield very good approximate solutions either.

In the present work, we approach the task of finding small but not necessarily minimal tile sets for a given 2D pattern by tailoring the basic partition-search framework of [3] towards this goal. Instead of a systematic branch-and-bound pruning and traversal of the complete search space, we apply a heuristic that attempts to optimize the order of the directions in which the space is explored. The new algorithm, denoted PS-H, is described in more detail below in Sect. 3.

It is well known in the heuristic optimization community [2,7] that when the runtime distribution of a randomized search algorithm has a large variance, it is with high probability more efficient to run several independent short runs (“restarts”) of the algorithm than a single long run. Correspondingly, we investigate the efficiency of the PS-H search method for a number of parallel executions ranging from 1 to 32, and note that indeed this number has a significant effect on the success rate of the algorithm in finding small tile sets. Also these results are discussed below in Sect. 3.

Given the inherently stochastic nature of the DNA self-assembly process, it is also of interest to assess the reliability of a given tile set, i.e. the probability of its error-free self-assembly to the desired target tiling. In Sect. 4 we introduce a method for estimating this quantity, based on Winfree’s analysis of the kinetic Tile Assembly Model [15]. We present empirical data on the reliability of tile sets found by the PS-BB and PS-H algorithms and find that also here the PS-H algorithm constitutes a significant improvement over the PS-BB method.

## 2 The PATS Problem and the PS-BB Algorithm

### 2.1 The Abstract Tile Assembly Model [11,15]

Let  $\mathcal{D} = \{N, E, S, W\}$  be the set of four functions  $\mathbb{Z}^2 \rightarrow \mathbb{Z}^2$  corresponding to the four cardinal directions. Let  $\Sigma$  be a finite set of *glue types* and  $s : \Sigma \times \Sigma \rightarrow \mathbb{N}$  a *glue strength* function such that  $s(\sigma, \sigma') > 0$  only if  $\sigma = \sigma'$ . A *tile type*  $t \in \Sigma^4$  is a quadruple  $(\sigma_N(t), \sigma_E(t), \sigma_S(t), \sigma_W(t))$  and a (*tile*) *assembly*  $\mathcal{A}$  is a partial

mapping from  $\mathbb{Z}^2$  to  $\Sigma^4$ . A *tile assembly system* (TAS)  $\mathcal{T} = (T, \mathcal{S}, s, \tau)$  consists of a finite set  $T$  of tile types, a *seed assembly*  $\mathcal{S}$ , a glue strength function  $s$  and a *temperature*  $\tau \in \mathbb{Z}^+$  (we use  $\tau = 2$ ).

Now consider a TAS  $\mathcal{T} = (T, \mathcal{S}, s, \tau)$ . Assembly  $\mathcal{A}$  *produces directly* assembly  $\mathcal{A}'$ , denoted  $\mathcal{A} \rightarrow_{\mathcal{T}} \mathcal{A}'$ , if there exists a site  $(x, y) \in \mathbb{Z}^2$  and a tile  $t \in T$  such that  $\mathcal{A}' = \mathcal{A} \cup \{(x, y), t\}$ , where the union is disjoint, and

$$\sum_D s(\sigma_D(t), \sigma_{D^{-1}}(\mathcal{A}(D(x, y)))) \geq \tau ,$$

where  $D$  ranges over those directions in  $\mathcal{D}$  for which  $\mathcal{A}(D(x, y))$  is defined. That is, a new tile can be adjoined to an assembly  $\mathcal{A}$  if the new tile shares a common boundary with tiles that bind it into place with total strength at least  $\tau$ .

Let  $\rightarrow_{\mathcal{T}}^*$  be the reflexive transitive closure of  $\rightarrow_{\mathcal{T}}$ . A TAS  $\mathcal{T}$  *produces* an assembly  $\mathcal{A}$  if  $\mathcal{S} \rightarrow_{\mathcal{T}}^* \mathcal{A}$ . Denote by  $\text{Prod } \mathcal{T}$  the set of all assemblies produced by  $\mathcal{T}$ . A TAS  $\mathcal{T}$  is *deterministic* if for any assembly  $\mathcal{A} \in \text{Prod } \mathcal{T}$  and for every  $(x, y) \in \mathbb{Z}^2$  there exists at most one  $t \in T$  such that  $\mathcal{A}$  can be extended with  $t$  at site  $(x, y)$ . Then the pair  $(\text{Prod } \mathcal{T}, \rightarrow_{\mathcal{T}}^*)$  forms a partially ordered set, which is a lattice if and only if  $\mathcal{T}$  is deterministic. The maximal elements in  $\text{Prod } \mathcal{T}$ , i.e. the assemblies  $\mathcal{A}$  for which there do not exist any  $\mathcal{A}'$  satisfying  $\mathcal{A} \rightarrow_{\mathcal{T}} \mathcal{A}'$ , are called *terminal assemblies*. Denote by  $\text{Term } \mathcal{T}$  the set of terminal assemblies of  $\mathcal{T}$ . If all *assembly sequences*  $\mathcal{S} \rightarrow_{\mathcal{T}} \mathcal{A}_1 \rightarrow_{\mathcal{T}} \mathcal{A}_2 \rightarrow_{\mathcal{T}} \dots$  terminate and  $\text{Term } \mathcal{T} = \{\mathcal{P}\}$  for some assembly  $\mathcal{P}$ , then  $\mathcal{T}$  *uniquely produces*  $\mathcal{P}$ .

### 2.2 The PATS Problem

Let the dimensions  $m$  and  $n$  be fixed. A mapping from  $[m] \times [n] \subseteq \mathbb{Z}^2$  onto  $[k]$  defines a *k-coloring* or a *k-colored pattern*. To build a given pattern, we start with boundary tiles in place for the west and south borders of the  $m$  by  $n$  rectangle and keep extending this assembly by tiles with strength-1 glues.

#### Definition 1 (Pattern self-Assembly Tile set Synthesis (PATS) [8])

**Given:** A *k-coloring*  $c : [m] \times [n] \rightarrow [k]$ .

**Find:** A *tile assembly system*  $\mathcal{T} = (T, \mathcal{S}, s, 2)$  such that

- P1. The tiles in  $T$  have glue strength 1.
- P2. The domain of  $\mathcal{S}$  is  $[0, m] \times \{0\} \cup \{0\} \times [0, n]$  and all the terminal assemblies have domain  $[0, m] \times [0, n]$ .
- P3. There exists a *tile coloring*  $d : T \rightarrow [k]$  such that each terminal assembly  $\mathcal{A} \in \text{Term } \mathcal{T}$  satisfies  $d(\mathcal{A}(x, y)) = c(x, y)$  for all  $(x, y) \in [m] \times [n]$ .

Finding minimal solutions (in terms of  $|T|$ ) to the PATS problem has been described as NP-hard in [8]. Without loss of generality, we consider only TASs  $\mathcal{T}$  in which every tile type participates in some terminal assembly of  $\mathcal{T}$ .

In the literature, the seed assembly of a TAS is often taken to be a single seed tile [11] whereas we consider an L-shaped seed assembly. The boundaries can always be self-assembled using  $m + n + 1$  different tiles with strength-2 glues, but we wish to make a clear distinction between the complexity of constructing the boundaries and the complexity of the 2D pattern itself.

### 2.3 The PS-BB Algorithm

The partition-search branch-and-bound (PS-BB) algorithm for the PATS problem proposed in [3], and based partly on ideas from [8], performs an exhaustive search in the lattice of partitions of the ambient rectangle  $[m] \times [n]$ . For each candidate partition  $P$ , the algorithm executes a polynomial-time test (details omitted in the present summary) to see if it is *constructible*, i.e. whether it can be produced by some deterministic TAS  $\mathcal{T}$ . If so, then the algorithm proceeds to consider coarsenings of  $P$  — these correspond to smaller tile systems — if not, then the algorithm backtracks. The search starts with the trivial partition which places each of the  $m \cdot n$  sites in different classes, corresponding to an initial tile set that contains a distinct tile type for each of the tile sites in  $[m] \times [n]$ .

Let us now review some of the basic notions of the PS-BB algorithm in more detail. In the following, a PATS instance is assumed to be given by a fixed  $k$ -colored pattern  $c : [m] \times [n] \rightarrow [k]$ .

**The Search Space.** Let  $X$  be the set of partitions of the set  $[m] \times [n]$ . Partition  $P$  is *coarser* than partition  $P'$  (or  $P'$  is a *refinement* of  $P$ ), denoted  $P \sqsubseteq P'$ , if

$$\forall p' \in P' : \exists p \in P : p' \subseteq p .$$

Now,  $(X, \sqsubseteq)$  is a partially ordered set, and in fact, a lattice. Note that  $P \sqsubseteq P'$  implies  $|P| \leq |P'|$ .

The coloring  $c$  induces a partition  $P(c) = \{c^{-1}(i) \mid i \in [k]\}$  of the set  $[m] \times [n]$ . In addition, since every (deterministic) solution  $\mathcal{T} = (T, \mathcal{S}, s, 2)$  of the PATS problem uniquely produces some assembly  $\mathcal{A}$ , we associate with  $\mathcal{T}$  a partition  $P(\mathcal{T}) = \{\mathcal{A}^{-1}(t) \mid t \in \mathcal{A}([m] \times [n])\}$ . Here,  $|P(\mathcal{T})| = |T|$  in case all tiles in  $T$  are used in the terminal assembly. Now the condition  $P\beta$  in the definition of PATS is equivalent to requiring that a TAS  $\mathcal{T}$  satisfies

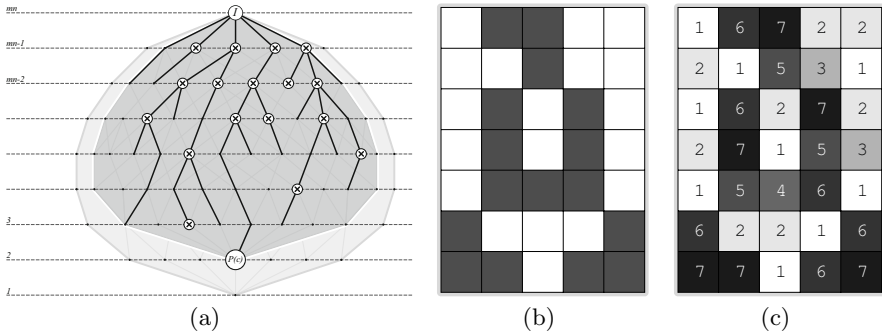
$$P(c) \sqsubseteq P(\mathcal{T}) .$$

A partition  $P \in X$  is *constructible* if  $P = P(\mathcal{T})$  for some deterministic TAS  $\mathcal{T}$  with properties  $P1$  and  $P2$ . Hence the PATS problem can be rephrased using partitions as the fundamental search space.

**Proposition 1.** *A minimal solution to the PATS problem corresponds to a partition  $P \in X$  such that  $P$  is constructible,  $P(c) \sqsubseteq P$ , and  $|P|$  is minimal.*

Schematically, the PS-BB algorithm performs an exhaustive top-to-bottom search in the lattice  $(X, \sqsubseteq)$  as illustrated in Fig. 1(a). The algorithm also involves several bounding heuristics for pruning the branches of the search, but discussion of these is omitted here for lack of space (see [3]).

For example, the 2-colored pattern in Fig. 1(b) defines a 2-part partition  $A$ . The 7-part partition  $M$  in Fig. 1(c) is a refinement of  $A$  ( $A \sqsubseteq M$ ) and in fact,  $M$  is constructible (see Fig. 2(b)) and corresponds to a minimal solution of the PATS problem defined by the pattern  $A$ .



**Fig. 1.** (a) The search lattice  $(X, \subseteq)$ . The search starts with the initial partition  $I$  of size  $|I| = mn$  (top) and considers the constructible partitions (crosses) in the upper sublattice of refinements of partition  $P(c)$  (bottom). (b) Partition  $A$ . (c) A partition  $M$  that is a refinement of  $A$  with  $|M| = 7$  parts.

**Determining Constructibility.** For lack of space, we shall omit the polynomial-time algorithm for testing whether a given partition  $P$  is constructible (see [3]), except for the mention of the following key notion.

**Definition 2.** Given a partition  $P$  of the set  $[m] \times [n]$ , a most general tile assignment (MGTA) is a function (“tile map”)  $f : P \rightarrow \Sigma^4$  such that

- A1.  $f$  is consistent: when sites in  $[m] \times [n]$  are assigned tile types according to  $f$ , any two adjacent sites have matching glues along their common side.
- A2.  $f$  is minimally constrained: any  $g : P \rightarrow \Sigma^4$  satisfying A1 satisfies also.<sup>1</sup>

$$f(p_1)_{D_1} = f(p_2)_{D_2} \implies g(p_1)_{D_1} = g(p_2)_{D_2} ,$$

for all partition classes  $p_1, p_2 \in P$  and directions  $D_1, D_2 \in \mathcal{D}$ .

As an illustration, a most general tile assignment  $f : I \rightarrow \Sigma^4$  for the initial partition  $I = \{\{a\} \mid a \in [m] \times [n]\}$  is presented in Fig. 2(a) and a MGTA for the partition of Fig. 1(c) in Fig. 2(b).

Given a partition  $P \in X$  and a tile map  $f : P \rightarrow \Sigma^4$ , tile map  $g : P \rightarrow \Sigma^4$  is obtained from  $f$  by merging glues  $a$  and  $b$  if for all  $(p, D) \in P \times \mathcal{D}$  we have

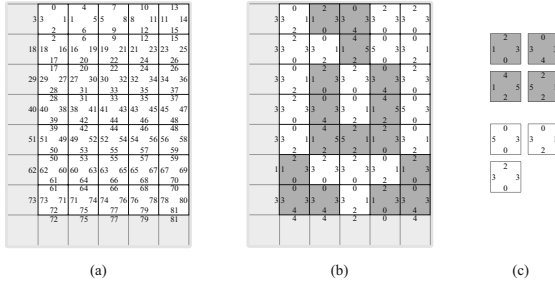
$$g(p)_D = \begin{cases} a, & \text{if } f(p)_D = b \\ f(p)_D, & \text{otherwise} \end{cases} .$$

A most general tile assignment for a partition  $P \in X$  can be found as follows. One starts with a map  $f_0 : P \rightarrow \Sigma^4$  that assigns to each tile edge a unique glue type. Next, one considers all pairs of adjacent sites in  $[m] \times [n]$  and makes their common sides matching by merging the corresponding glues. This process generates a sequence of tile maps  $f_0, f_1, f_2, \dots, f_N = f$  and ends after  $N \leq 2mn$  steps.

**Lemma 1.** [3] The above algorithm generates a most general tile assignment.

<sup>1</sup> For brevity we write  $f(p)_D$  instead of  $\sigma_D(f(p))$ .





**Fig. 2.** (a) A MGTA for the constructible initial partition  $I$  (with a seed assembly in place). (b) Finished assembly for the pattern from Fig. 1(b). The tile set to construct this assembly is given in (c).

### 3 A New Algorithm for Small Tile Sets

Whereas the pruning procedures of the PS-BB algorithm try to reduce the size of the search space in a “balanced” way, our new PS-H algorithm attempts to “greedily” optimize the order in which the coarsenings of a partition are explored, in the hope of being directly lead to close-to-optimal solutions. Such opportunism may be expected to pay off in case the success probability of the greedy exploration is sufficiently high, and the process is restarted sufficiently often, or equivalently several runs are explored in parallel.

The basic heuristic idea is to try to minimize the effect that a merge operation has on other partition classes than those which are combined. This can be achieved by preferring to merge classes already having as many common glues as possible. In this way one hopes to extend the number of steps the search takes before it runs into a conflict. For example, when merging classes  $p_1$  and  $p_2$  such that  $f(p_1)_N = f(p_2)_N$  and  $f(p_1)_E = f(p_2)_E$ , the glues on the W and S edges of all other classes are unaffected. This way, the search avoids proceeding to a partition which is not constructible after the merge operation is completed. Secondly, we prefer merging classes which already cover a large number of sites in  $[m] \times [n]$ . That is, one tries to grow a small number of large classes instead of growing all the classes at an equal rate.

**Definition 3.** Given a partition  $P$  and a MGTA  $f$  for  $P$ , the number of common glues between classes  $p, q \in P$  is defined by  $G : P \times P \rightarrow \{0, 1, 2, 3, 4\}$ ,

$$G(p, q) = \sum_{D \in \mathcal{D}} g(f(p)_D, f(q)_D) ,$$

where  $g(\sigma, \sigma') = 1$  if  $\sigma = \sigma'$  and 0 otherwise, for  $\sigma, \sigma' \in \Sigma$ .

Except for the bounding function, the PS-BB algorithm allows an arbitrary ordering  $\{p_i, q_i\}, i = 1, \dots, N$ , for the children (coarsenings)  $P[p_i, q_i]$  of a constructible partition  $P$ .<sup>2</sup> In the PS-H algorithm, we choose the ordering using the following heuristic. First form the set

<sup>2</sup>  $P[p, q]$  denotes the partition obtained from  $P$  by merging classes  $p, q \in P$  into one.

$$H := \{\{p, q\} \mid p, q \in P, p \neq q, \exists k \in P(c) : p, q \subseteq k\}$$

of class pairs of same color. Then, repeat the following process until  $H$  is empty.

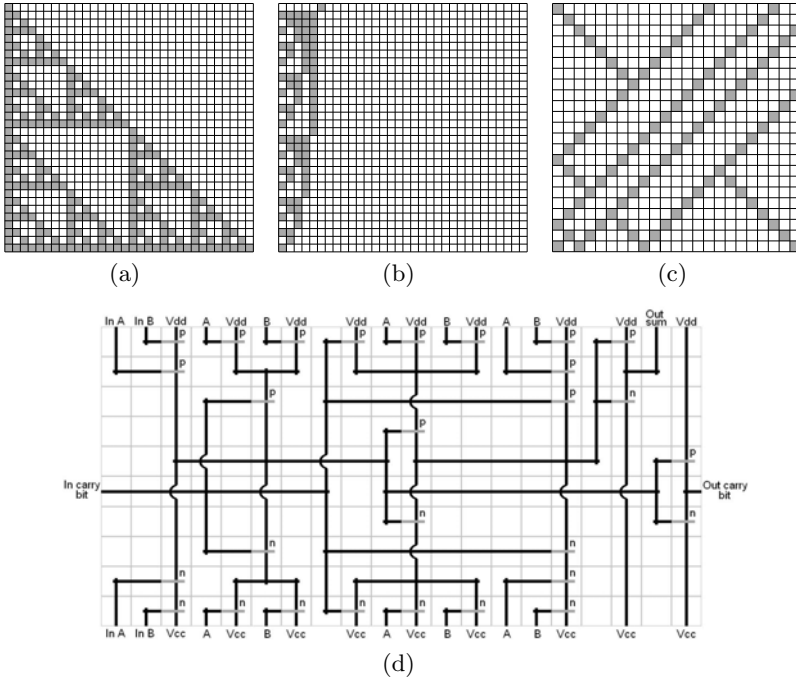
- H1. Set  $K := H$ .
- H2. Optimize the number of common glues:  
 $K := \{\{p, q\} \in K \mid G(p, q) \geq G(u, v) \text{ for all } \{u, v\} \in K\}$ .
- H3. Optimize the size of the larger class:  
 $K := \{\{p, q\} \in K \mid \max\{|p|, |q|\} \geq \max\{|u|, |v|\} \text{ for all } \{u, v\} \in K\}$ .
- H4. Optimize the size of the smaller class:  
 $K := \{\{p, q\} \in K \mid \min\{|p|, |q|\} \geq \min\{|u|, |v|\} \text{ for all } \{u, v\} \in K\}$ .
- H5. Pick some pair  $\{p, q\} \in K$  at random and visit the partition  $P[p, q]$ .
- H6. Remove  $\{p, q\}$  from  $H$ :  $H := H \setminus \{\{p, q\}\}$ .

The PS-H algorithm also omits the pruning process utilized by the PS-BB algorithm. That way, it aims to get to the small solutions quickly by reducing the computational resources used in a single merge operation.

Since step H5 above leaves room for randomization, the PS-H algorithm performs differently with different seeds. While some of the randomized runs may lead to small solutions quickly, others may get sidetracked into worthless expanses of the solution space. We make the best of this situation by running several instances of the algorithm in parallel, or equivalently, restarting the search several times with a different random seed. The notation PS-H<sub>*n*</sub> denotes the heuristic partition search algorithm with *n* parallel search threads. The solution of the PS-H<sub>*n*</sub> algorithm is the smallest solution found by any of the *n* threads.

**Results.** Our implementation of the PS-H algorithm is based on the DFS implementation of the PS-BB algorithm used in [3], and we provide results on the PS-H<sub>*n*</sub> algorithm for  $n = 1, 2, 4, 8, 16$  and 32. We consider several different finite 2-colored input patterns, two of which are classical examples of structured patterns: the discrete Sierpinski triangle (Fig. 3(a)), and the binary counter (Fig. 3(b)). Furthermore, we introduce a 2-colored “tree” pattern of size  $23 \times 23$  (Fig. 3(c)) as well as a 15-colored pattern of size  $20 \times 10$  based on a CMOS full adder design (Fig. 3(d), [1]). The Sierpinski triangle and binary counter patterns are known to have a minimal solution of four tiles, while the minimal solutions for the tree and the full adder patterns are unknown.

Figure 4 presents the evolution of the “current best solution” as a function of time for the (a)  $32 \times 32$  and (c)  $64 \times 64$  Sierpinski patterns. To allow fair comparison, Figs. 4(b) and 4(d) present the same data with respect to the total processing time taken by all the parallelly running instances. The experiments were repeated 21 times and the median of the results is depicted. In 37% of all the runs conducted, the PS-H algorithm is able to find the optimal 4-tile solution for the  $32 \times 32$  Sierpinski pattern in less than 30 seconds. The similar percentage for the  $64 \times 64$  Sierpinski pattern is 34% in one hour. Remarkably, the algorithm performs only from 1030 to 1035 and from 4102 to 4107 merge steps before arriving at the optimal solution for the  $32 \times 32$  and  $64 \times 64$  patterns,



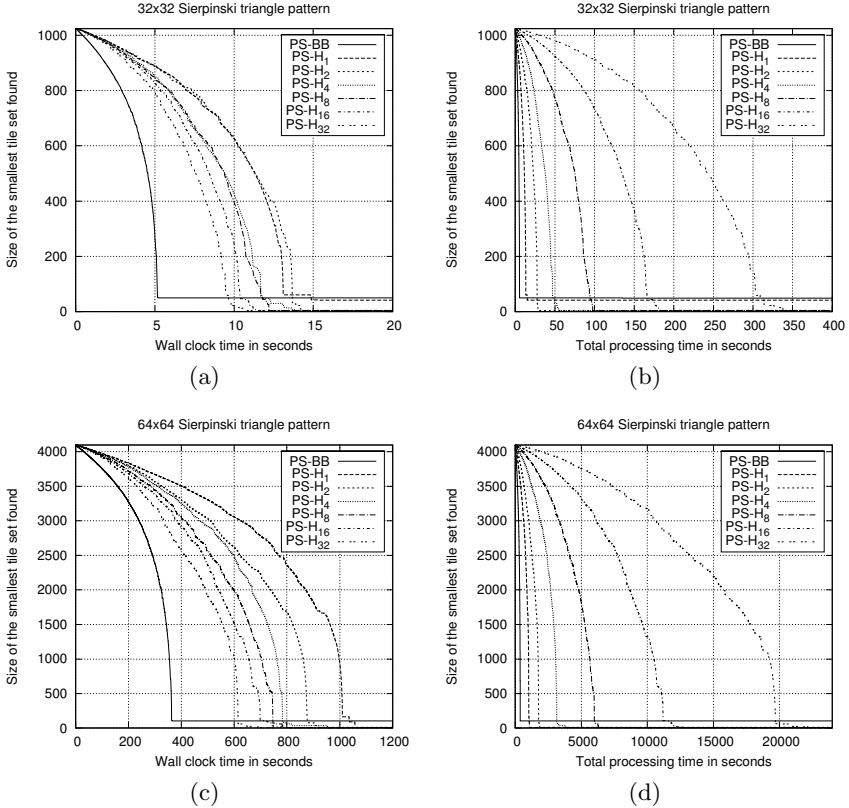
**Fig. 3.** (a) The  $32 \times 32$  Sierpinski triangle pattern. (b) The  $32 \times 32$  binary counter pattern. (c) The  $23 \times 23$  “tree” pattern. (d) A CMOS full adder design that induces a 15-color  $20 \times 10$  pattern.

respectively. In other words, the search rarely needs to backtrack. In contrast, the smallest solutions found by the PS-BB algorithm are 42 tiles, reached after  $1.4 \cdot 10^6$  merge steps, and 95 tiles, reached after  $5.9 \cdot 10^6$  merge steps.

In Fig. 5 we present the corresponding results for the  $32 \times 32$  binary counter and the  $23 \times 23$  tree patterns. The size of the smallest solutions found by the PS- $H_{32}$  algorithm were 20 (cf. 307 by PS-BB) and 25 (cf. 192 by PS-BB) tiles, respectively. In the case of the tree pattern, the parallelization brings significant advantage over a single run. Finally, Figs. 6(a)–6(b) show the results for the  $20 \times 10$  15-color CMOS full adder pattern. In this case, the improvement over the previous PS-BB algorithm is less clear. The PS- $H_{32}$  algorithm is able to find a solution of 58 tiles, whereas the PS-BB algorithm gives a solution of 69 tiles.

## 4 Reliability of Tile Sets

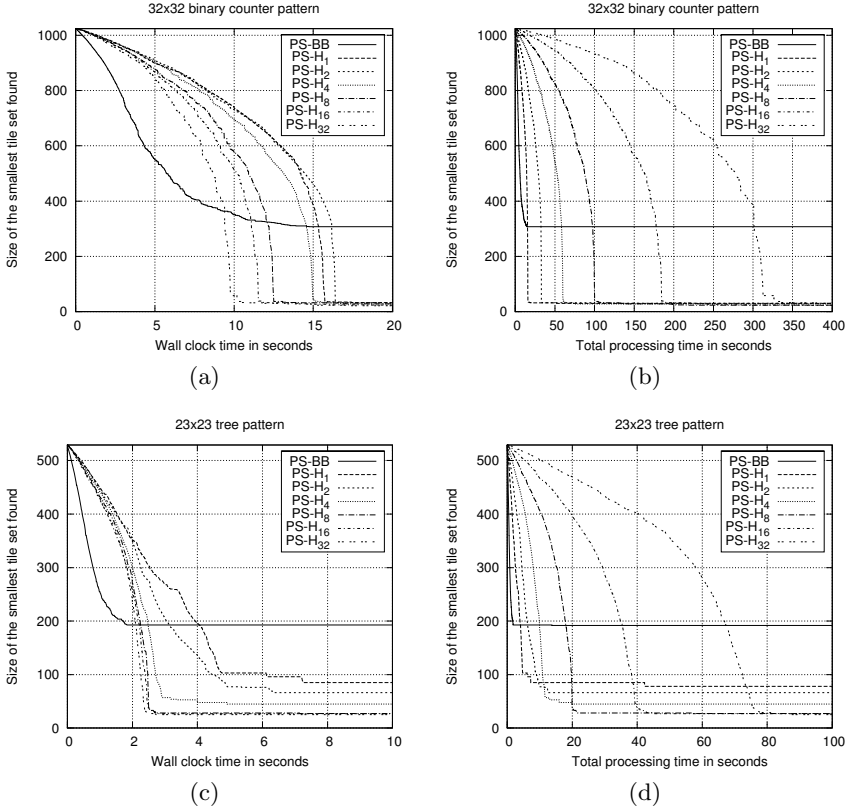
**The Kinetic Tile Assembly Model.** In this section, we assess the reliability of the tile sets produced by the PS-BB and PS-H algorithms, using the kinetic Tile Assembly Model (kTAM), which has been proposed by Winfree [15] as a kinetic counterpart of the aTAM.



**Fig. 4.** Evolution of the smallest tile set as a function of time. The time axes measure (a), (c) wall clock time and (b), (d) wall clock time  $\times$  the number of parallel instances.

The kTAM simulates two types of reactions: association of tiles to the assembly (forward reaction), and dissociation (reverse reaction). In the first type, any tile can attach to the assembly at any position, even if only a weak bond is formed; the rate of this reaction,  $r_f$ , is proportional to the concentration of free tiles in the solution. In the second type, any tile can detach from the assembly, with rate  $r_{r,b}$  where  $b \in \{0, \dots, 4\}$ , which is exponentially correlated with the total strength of the bonds between the tile and the assembly. Thus, tiles which are connected to the assembly by fewer or weaker bonds, i.e. incorrect “sticky end” matches, are more prone to dissociation than those which are strongly connected by several bonds (well paired sticky end sequences).

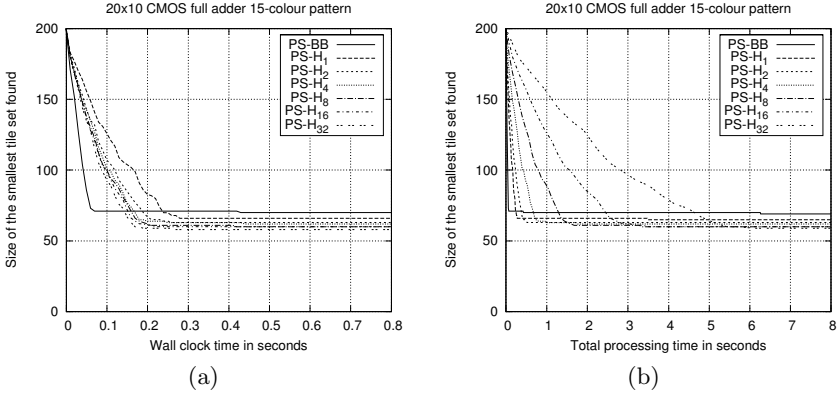
In order to easily represent and scale the system, the free parameters involved in the formulas of  $r_f$  and  $r_{r,b}$  are re-distributed into just two dimensionless parameters,  $G_{mc}$  and  $G_{se}$ . The first is dependent on the initial tile concentration, while the second is dependent on the assembly temperature:  $r_f = \hat{k}_f e^{-G_{mc}}$  and  $r_{r,b} = \hat{k}_f e^{-bG_{se}}$  where  $\hat{k}_f = e^3 k_f$  is adjusted in order to take into consideration possible entropic factors, such as orientation or location of the tiles.



**Fig. 5.** Evolution of the smallest tile set as a function of time. The time axes measure (a), (c) wall clock time and (b), (d) wall clock time  $\times$  the number of parallel instances.

**Computing the Reliability of a Tile Set.** The probability of errors in the assembly process can be made arbitrarily low, at the cost of reduced speed, by choosing appropriate physical conditions [15]. However, we would like to be able to compare the error probability of different tile sets producing the same finite pattern, under the same physical conditions. Given the amount of time the assembly process is allowed to take, we define the *reliability of a tile set* to be the probability that the assembly process of the tile system in question completes without any incorrect tiles being present in the terminal configuration. In the following, we present a method for computing the reliability of a tile set, based on Winfree’s analysis of the kTAM and the notion of *kinetic trapping* in [15].

We call the west and south edges of a tile its *input edges*. First, we derive the probability of the correct tile being frozen at a particular site under the condition that the site already has correct tiles on its input edges. Let  $M_{ij}^1$  and  $M_{ij}^2$  be the number of tile types having one mismatching and two mismatching glue types, respectively, between them and the correct tile type for site  $(i, j) \in [m] \times [n]$ .



**Fig. 6.** Evolution of the smallest tile set as a function of time. The time axes measure (a) wall clock time and (b) wall clock time  $\times$  the number of parallel instances.

Now, for a deterministic tile set  $T$ , the total number of tiles is  $|T| = 1 + M_{ij}^1 + M_{ij}^2$  for all  $i$  and  $j$ . Given that a site has correct tiles on its input edges, a tile is correct for that site if and only if it has two matches on its input edges.

In what follows, we assume correct tiles are attached at sites  $(i - 1, j)$  and  $(i, j - 1)$ . The model for kinetic trapping [15] gives four distinct cases in the situation preceding the site  $(i, j)$  being frozen by further growth: (E) An empty site, with “off-rate”  $|T|r_f$ . (C) The correct tile, with off-rate  $r_{r,2}$ . (A) A tile with one match, with off-rate  $r_{r,1}$ . (I) A tile with no matches, with off-rate  $r_{r,0}$ . Additionally, we have two sink states FC and FI, which represent frozen correct and frozen incorrect tiles, respectively. The rate of a site being frozen is equal to the rate of growth  $r^* = r_f - r_{r,2}$ . Let  $p_S(t)$  denote the probability of the site being in state  $S$  after  $t$  seconds for all  $S \in \{E, C, A, I, FC, FI\}$ . To compute the frozen distribution, we write rate equations for the model of kinetic trapping:

$$M\mathbf{p}(t) := \begin{bmatrix} -|T|r_f & r_{r,2} & r_{r,1} & r_{r,0} & 0 & 0 \\ r_f & -r_{r,2} - r^* & 0 & 0 & 0 & 0 \\ M_{ij}^1 r_f & 0 & -r_{r,1} - r^* & 0 & 0 & 0 \\ M_{ij}^2 r_f & 0 & 0 & -r_{r,0} - r^* & 0 & 0 \\ 0 & r^* & 0 & 0 & 0 & 0 \\ 0 & 0 & r^* & r^* & 0 & 0 \end{bmatrix} \begin{bmatrix} p_E(t) \\ p_C(t) \\ p_A(t) \\ p_I(t) \\ p_{FC}(t) \\ p_{FI}(t) \end{bmatrix} = \dot{\mathbf{p}}(t) ,$$

where  $\mathbf{p}(0) = [1 \ 0 \ 0 \ 0 \ 0 \ 0]^T$ . To compute the probability of the site being frozen with the correct tile,  $p_{FC}(\infty)$ , we make use of the steady state of the related flow problem [15]:

$$M\mathbf{p}(\infty) = [1 \ 0 \ 0 \ 0 \ p_{FC}(\infty) \ p_{FI}(\infty)]^T = \dot{\mathbf{p}}(\infty) ,$$

which gives us a system of linear equations. This system has a single solution

$$p_{\text{FC}}(\infty) = \frac{\frac{1}{r^* + r_{r,2}}}{\frac{1}{r^* + r_{r,2}} + \frac{M_{ij}^1}{r^* + r_{r,1}} + \frac{M_{ij}^2}{r^* + r_{r,0}}} = \Pr(C_{i,j} \mid C_{i-1,j} \cap C_{i,j-1}) ,$$

where  $C_{i,j}$  denotes the correct tile being frozen on site  $(i, j)$ .

The assembly process can be thought of as a sequence of tile addition steps  $(a_1, a_2, \dots, a_N)$  where  $a_k = (i_k, j_k)$ ,  $k = 1, 2, \dots, N$  denotes a tile being frozen on site  $(i_k, j_k)$ . Due to the fact that the assembly process of the tile systems we consider proceeds uniformly from south-west to north-east,  $\{(i_k - 1, j_k), (i_k, j_k - 1)\} \subseteq \{a_1, a_2, \dots, a_{k-1}\}$  for all  $a_k = (i_k, j_k)$ . We assume that tiles elsewhere in the configuration do not affect the probability. Now we can compute the probability of a finite-size pattern of size  $N$  assembling without any errors, i.e. the reliability of that pattern:

$$\begin{aligned} \Pr(\text{correct pattern}) &= \Pr(C_{a_1} \cap C_{a_2} \cap \dots \cap C_{a_N}) \\ &= \Pr(C_{a_1})\Pr(C_{a_2} \mid C_{a_1}) \dots \Pr(C_{a_N} \mid C_{a_1} \cap C_{a_2} \cap \dots \cap C_{a_{N-1}}) \\ &= \prod_{i,j} \Pr(C_{i,j} \mid C_{i-1,j} \cap C_{i,j-1}) . \end{aligned}$$

We have computed the probability in terms of  $G_{\text{mc}}$  and  $G_{\text{se}}$ . Given the desired assembly speed, we want to minimize the error probability by choosing values for  $G_{\text{mc}}$  and  $G_{\text{se}}$  appropriately. If the assembly process is allowed to take  $t$  seconds, the needed assembly speed for an  $m \times n$  pattern is approximately  $r^* = \frac{\sqrt{m^2+n^2}}{t}$ .

$$\Pr(C_{i,j} \mid C_{i-1,j} \cap C_{i,j-1}) = \frac{\frac{1}{r^* + r_{r,2}}}{\frac{1}{r^* + r_{r,2}} + \frac{M_{ij}^1}{r^* + r_{r,1}} + \frac{M_{ij}^2}{r^* + r_{r,0}}} \approx \frac{1}{1 + M_{ij}^1 \frac{r^* + r_{r,2}}{r^* + r_{r,1}}} .$$

For small error probability and  $2G_{\text{se}} > G_{\text{mc}} > G_{\text{se}}$ ,

$$\Pr(\neg C_{i,j} \mid C_{i-1,j} \cap C_{i,j-1}) \approx M_{ij}^1 \frac{r^* + r_{r,2}}{r^* + r_{r,1}} \approx M_{ij}^1 e^{-(G_{\text{mc}} - G_{\text{se}})} =: M_{ij}^1 e^{-\Delta G} .$$

From  $r^* = r_f - r_{r,2} = \hat{k}_f(e^{-G_{\text{mc}}} - e^{-2G_{\text{se}}})$  we can derive

$$G_{\text{se}} = -\frac{1}{2} \log\left(e^{-G_{\text{mc}}} - \frac{r^*}{\hat{k}_f}\right) .$$

Now we can write  $\Delta G$  as a function of  $G_{\text{mc}}$ :

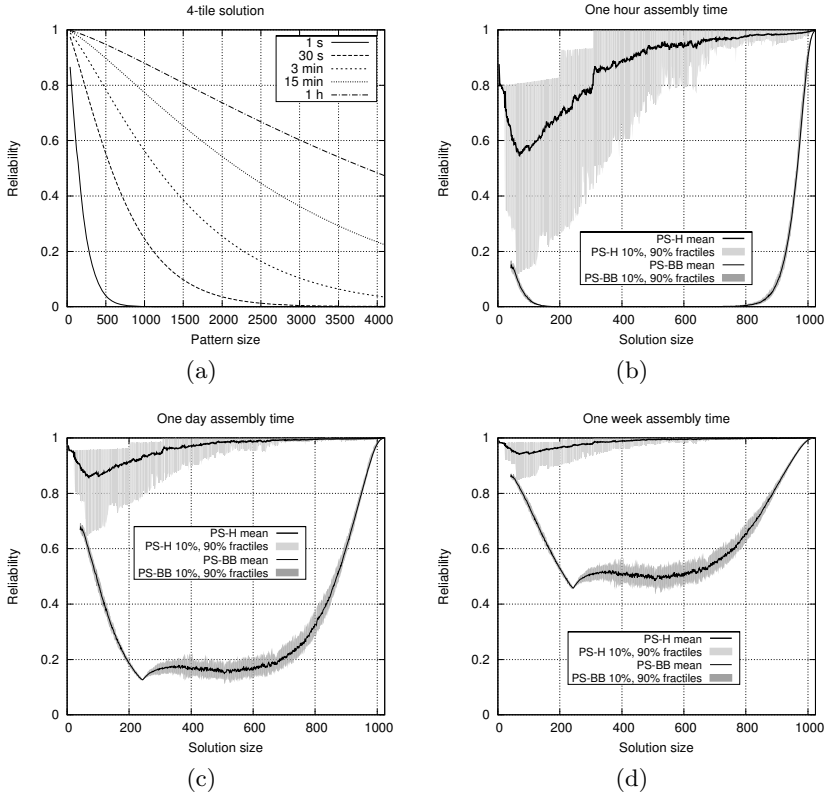
$$\Delta G(G_{\text{mc}}) = G_{\text{mc}} - G_{\text{se}} = G_{\text{mc}} + \frac{1}{2} \log\left(e^{-G_{\text{mc}}} - \frac{r^*}{\hat{k}_f}\right) .$$

We find the maximum of  $\Delta G$  and the minimal error probability by differentiation:

$$G_{\text{mc}} = -\log\left(2\frac{r^*}{\hat{k}_f}\right) .$$

Thus, if the assembly time is  $t$  seconds, the maximal reliability is achieved at

$$G_{\text{mc}} = -\log\left(2\frac{\sqrt{m^2+n^2}}{t\hat{k}_f}\right) , \quad G_{\text{se}} = -\frac{1}{2} \log\left(\frac{\sqrt{m^2+n^2}}{t\hat{k}_f}\right) .$$



**Fig. 7.** (a) Reliability of the minimal tile set as a function of pattern size for the Sierpinski pattern, using several different assembly times. (b)–(d) Reliability of the solutions for the  $32 \times 32$  Sierpinski pattern found by the PS-H and PS-BB algorithms, allowing assembly time of one hour, one day and one week.

**Results.** In the following, we present results on computing the reliability of tile sets using the above method. We assume the assembly process takes place in room temperature (298 K). As a result, we use the value  $k_f = A_f e^{-E_f/RT} \approx 6 \cdot 10^5$  /M/sec for the forward reaction rate.

Figure 7(a) shows the reliability of the 4-tile solution to the Sierpinski pattern as a function of pattern size, using five distinct assembly times. As is expected, the longer the assembly time, the better the reliability.

We also applied the method for computing the reliability to the tile sets found by the partition search algorithms. Our results show that the heuristic described in Sect. 3 improves not only the size of the tile sets found, but also the reliability of those tile sets. This can be easily understood by considering the following: the reliability of a tile set is largely determined by the number of tile types that



have the same glue as some other tile type on either one of their input edges. Since the heuristic prefers merging class pairs with common glues, it reduces the number of such tile types effectively.

Figures 7(b)–7(d) present the reliability of the tile sets found by the PS-H and PS-BB algorithms for the  $32 \times 32$  Sierpinski triangle pattern, using assembly times of one hour, one day (24 hours) and one week. The runs were repeated 100 times; the mean reliability of each tile set size as well as the 10th and 90th percentiles are shown.

As for reliability, we expect a large set of runs of the PS-BB algorithm to produce a somewhat decent sample of all the possible tile sets for a pattern. Based on this, large and small tile sets seem to have a high reliability while medium-size tile sets are clearly more unreliable on average. This observation reduces the problem of finding reliable tile sets back to the problem of finding small tile sets. However, it is important to note that artifacts of the algorithm may have an effect on the exact reliability of the tile sets found.

## 5 Conclusion

We presented a new algorithm, PS-H, for addressing the problem of finding small tile sets that have a high probability of self-assembling a given target pattern. Our results show that for most patterns, the new algorithm is able to find significantly smaller solutions in a reasonable amount of time compared to the earlier PS-BB algorithm. Also the reliability of the tile sets produced by the PS-H algorithm clearly exceeds that of the tile sets produced by the PS-BB algorithm.

In work not presented here for lack of space, we have explored the PATS problem also using the artificial intelligence technique of answer set programming (ASP) [5]. ASP is a declarative logic programming paradigm for solving difficult combinatorial search problems. In ASP, a problem is described as a logic program, and an answer set solver is then used to compute stable models (answer sets) for the logic program. Using the answer set solver SMOODELS [13], we considered finding minimal solutions for several patterns such as the Sierpinski triangle, the binary counter, and the full adder. Based on our results, we conclude that the ASP approach performs rather well when considering patterns with a small minimal solution. For example, the SMOODELS system was able to find the minimal four tile solutions even for the  $100 \times 100$  Sierpinski triangle and the  $100 \times 100$  binary counter patterns. However, for patterns with a larger minimal solution, the running time seems to increase dramatically.

## References

1. Czeizler, E., Lempiäinen, T., Orponen, P.: A design framework for carbon nanotube circuits affixed on DNA origami tiles. In: Proc. 8th Ann. Conf. Foundations of Nanoscience, pp. 186–187 (2011)
2. Gomes, C.P., Selman, B.: Algorithm portfolios. *Artif. Intell.* 126, 43–62 (2001)

3. Göös, M., Orponen, P.: Synthesizing minimal tile sets for patterned DNA self-assembly. In: Sakakibara, Y., Mi, Y. (eds.) DNA 16 2010. LNCS, vol. 6518, pp. 71–82. Springer, Heidelberg (2011)
4. Kim, K.N., Sarveswaran, K., Mark, L., Lieberman, M.: DNA origami as self-assembling circuit boards. In: Calude, C.S., Hagiya, M., Morita, K., Rozenberg, G., Timmis, J. (eds.) UC 2010. LNCS, vol. 6079, pp. 56–68. Springer, Heidelberg (2010)
5. Lifschitz, V.: What is answer set programming? In: Proc. 23rd Natl. Conf. Artificial Intelligence, pp. 1594–1597 (2008)
6. Liu, W., Zhong, H., Wang, R., Seeman, N.C.: Crystalline two-dimensional DNA-origami arrays. *Angewandte Chemie International Edition* 50(1), 264–267 (2011)
7. Luby, M., Sinclair, A., Zuckerman, D.: Optimal speedup of Las Vegas algorithms. *Information Processing Letters* 47(4), 173–180 (1993)
8. Ma, X., Lombardi, F.: Synthesis of tile sets for DNA self-assembly. *IEEE Trans. CAD of Integrated Circuits and Systems* 27, 963–967 (2008)
9. Maune, H.T., Han, S., Barish, R.D., Bockrath, M., Goddard III, W.A., Rothemund, P.W.K., Winfree, E.: Self-assembly of carbon nanotubes into two-dimensional geometries using DNA origami templates. *Nature Nanotechnology* 5, 61–66 (2010)
10. Rajendran, A., Endo, M., Katsuda, Y., Hidaka, K., Sugiyama, H.: Programmed two-dimensional self-assembly of multiple DNA origami jigsaw pieces. *ACS Nano* 5(1), 665–671 (2011)
11. Rothemund, P.W.K., Winfree, E.: The program-size complexity of self-assembled squares. In: Proc. 32nd Ann. ACM Theory of Computing, pp. 459–468 (2000)
12. Rothemund, P.W.K.: Folding DNA to create nanoscale shapes and patterns. *Nature* 440, 297–302 (2006)
13. Syrjänen, T., Niemelä, I.: The Smodels system. In: *Logic Programming and Non-monotonic Reasoning*, pp. 434–438. Springer, Heidelberg (2001)
14. Winfree, E., Liu, F., Wenzler, L.A., Seeman, N.C.: Design and self-assembly of two-dimensional DNA crystals. *Nature* 394 (1998)
15. Winfree, E.: *Simulations of Computing by Self-Assembly*. Technical Report CSTR 1998.22, California Institute of Technology (1998)
16. Yan, H., Park, S.H., Finkelstein, G., Reif, J.H., LaBean, T.H.: DNA-templated self-assembly of protein arrays and highly conductive nanowires. *Science* 301 (2003)

# Multivalent Random Walkers — A Model for Deoxyribozyme Walkers

Mark J. Olah and Darko Stefanovic

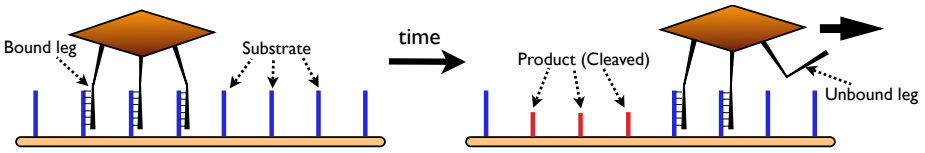
Department of Computer Science, University of New Mexico  
MSC01 1130, 1 University of New Mexico, Albuquerque, NM 87131

**Abstract.** We propose a stochastic model for molecular transport at the nanoscale that describes the motion of two-dimensional molecular assemblies called multivalent random walkers (MVRWs). This walker model is an abstract description of the motion of multipedal molecular assemblies, called *molecular spiders*, which use deoxyribozyme legs to move over a surface covered with substrate DNA molecules, cleaving them to produce shorter product DNA molecules as they go. In this model a walker has a rigid inert body and several flexible enzymatic legs. A walker moves over a surface of fixed chemical sites. Each site has one of several molecular species displayed, and walker legs can bind to and unbind from these sites to move over the surface. Additionally, the enzymatic activity of the legs allows them to catalyze irreversible chemical changes to the sites, thereby permanently modifying the state of the surface. We describe a MVRW system as a continuous-time Markov process, where all state transitions in the process correspond to chemical reactions of the legs with the sites. We model the kinetics of the leg reactions by considering the constrained diffusion of the walker body and unattached leg. Through kinetic Monte Carlo simulations, we show that the irreversibility of the enzymatic action of the legs can bias the motion of walkers and cause them to move superdiffusively over significant distances.

## 1 Introduction

Nature at the nanoscale is different from our familiar macroscopic experience in many ways, the most fundamental of which is the stochastic character of motion and events. At this scale, all objects experience random collisions with molecules that transfer significant energy, effectively randomizing momentum and leading to diffusive motion. Diffusive motion is often not desirable as it becomes a limiting factor in the transfer of material and information in chemical computational systems. However, nanoscale walkers have the potential to move in purposeful, directed ways by expending energy to bias their otherwise diffusive motion, thus providing a mechanism for superdiffusive motion.

Recently a new class of molecular walker based on DNA has been synthetically constructed. These *molecular spiders* [11] consist of a rigid, inert body and several deoxyribozyme (i.e., catalytic single-stranded DNA) legs that act as enzymes and attach to and cleave complementary single-stranded DNA substrates (at a sepecific ribonucleotide impurity). When the substrates are arrayed as nanoscale tracks and paths on a surface the walker can move along such tracks by binding, cleaving, and unbinding from the



**Fig. 1.** A molecular spider moves over a surface covered with fixed chemical substrate sites as legs bind and unbind to the sites

track sites [8]. As shown in Fig. 1, when the legs enzymatically modify a bound site by cleaving the substrate they leave behind a shorter product DNA sequence. The product remains complementary to the lower part of the leg. Thus, the legs can walk back over the product sites, albeit at a different rate than that for the substrates, and they can no longer modify the product sites.

In order to understand how molecular spiders move, we have developed the multivalent random walker (MVRW) model. The model describes spiders in a 2-dimensional environment of chemical sites. The motion of the spiders is modeled as a continuous-time Markov process, where each transition in the Markov process corresponds to a chemical reaction between a leg and a surface-bound site.

In this work we describe the model in detail, and briefly discuss our Monte Carlo simulation methods. We explain that when there is a residency-time bias between modified and unmodified sites, the walker motion is biased in the direction of unmodified sites. Through simulation we show that this bias causes the walker to move superdiffusively, even in opposition to a force.

## 2 The Multivalent Random Walker Model

At the single-molecule level, chemical kinetics are stochastic in nature. Each individual reaction can be viewed as a transition between two different chemical states of the system as a whole. Accordingly, chemical systems at the single-molecule level can be modeled as continuous-time stochastic processes [9]. A key assumption in such stochastic models is that the system reaches a physical equilibrium (i.e., it is well mixed) in between successive chemical reactions. This makes the system Markovian, and it makes determining the rates of chemical reactions tractable, as the exact position and momentum of particles do not need to be part of the system state, and the state space of the system remains discrete. In the 1970's Gillespie popularized the use of Monte Carlo methods for numerical simulation of stochastic chemical kinetics [3].

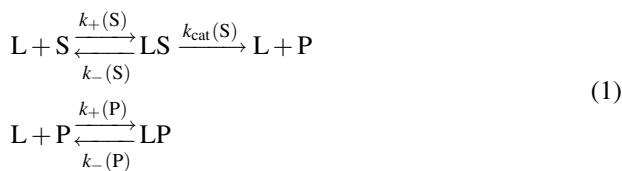
Inspired by this approach to chemical kinetics, the MVRW model describes the motion of the molecular spider as a discrete-state, continuous-time Markov process where each transition corresponds to a chemical reaction. In our case, this is a reaction of a leg binding, unbinding, or cleaving sites on the surface, but under the restrictions imposed by the attachment of the legs to a common body. In between reactions, the walker and its legs are assumed to reach a physical equilibrium over all feasible positions. By computing the distribution of the spider's body location after each step, we can accurately model the chemical reactions and how their rates are affected by the spatial constraints imposed by the spiders' geometry and the pattern of sites on the surface.

## 2.1 The State Space of the Walker and the Environment

In the MVRW model, the state of the Markov process is defined by the state of the walker and the state of the environment. Walkers are two-dimensional (2D), with a point body to which are attached  $k$  flexible legs. Each leg has length  $\ell$  and a reactive site at the end called the *foot*. The walkers move in an environment of fixed chemical sites. The *environment* is defined by a (countable) set  $S \subset \mathbb{R}^2$  of sites and a finite set  $\Sigma$  of species. Each site has a single species associated with it, but the species can be changed by the action of the walker legs. Thus, the state of the environment is defined by a mapping  $\pi : S \rightarrow \Sigma$  that assigns a species to each site. The state of the walker is completely described by the state of its  $k$  feet. A foot is either attached to a site in  $S$  or is detached. No two feet may be attached to the same site. The state of the walker is represented by the number of detached legs  $0 \leq d \leq k$ , and the set  $A \subset S$  of attached sites. Thus the state of the MVRW system is defined by the triple  $(\pi, d, A)$ .

## 2.2 State Transitions

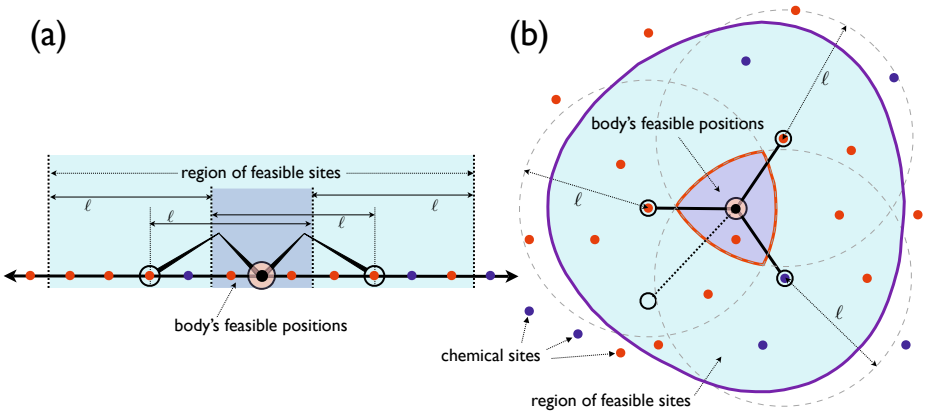
There are three types of state transitions corresponding to the three types of chemical reactions that can take place: binding (association), unbinding (dissociation), and catalytic transformation (cleavage). While the model can accommodate more general leg-site chemistries, we focus on the chemistry of the deoxyribozyme-based molecular spiders. In this system, there are two species  $\Sigma = \{S, P\}$ , a substrate and a product (cleaved) oligonucleotide. A leg (L) can reversibly bind to each species to form a leg-substrate (LS) or leg-product (LP) complex. Additionally, an LS complex can undergo catalysis, transforming the leg into a product before eventually unbinding. These reactions and the relevant rates are defined in Eq. 1. Note that we take  $k_{\text{cat}} = k_{\text{cat}}(S)$  to encompass the rate of cleavage as well as the subsequent dissociation, and we assume this process is irreversible.



Of the three types of state transitions, dissociation and cleavage are both unimolecular reactions and, as in the Gillespie model of chemical kinetics, each individual LS or LP pair will dissociate or cleave according to the rates  $k_-(S)$ ,  $k_-(P)$ , and  $k_{\text{cat}}$ . However, the association reactions are more complicated as they are bimolecular and their propensity depends on the likelihood of the leg being proximate to the chemical site, so that it may bind. This likelihood, in turn, depends on the position of the body and the unattached legs.

## 2.3 The Equilibrium Body Distribution

The states of the environment and the walker have been defined to capture only the parts of the system that remain fixed in between reaction events, but that change after



**Fig. 2.** The feasible body positions  $F$  for spiders in 1D (a) and 2D (b) are indicated in dark blue. We assume all legs have a maximum length  $\ell$  and the body positions cannot violate these constraints. In light blue we show the region of feasible sites. These are the sites that can be reached by an unattached leg from some feasible body position.

a reaction. Notice that the states are discrete, and that the body position is not part of the system state. In this way, the state transitions correspond directly to the chemical reactions and not to the physical motion of the walker body and legs. We assume that non-reactive processes, such as solvent collisions and molecular vibrations, occur on much faster timescales than the chemical reactions so that they come to an equilibrium in between state transitions.

First, let us consider the diffusion of the body. In between reaction events, the body will move in a constrained diffusion. Let random variable  $\mathbf{B}$  give the 2D coordinates of the spider's body. We assume the legs are flexible with a maximum length  $\ell$  and do not become tangled. Thus, the body will be constrained to be within distance  $\ell$  from each site with an attached foot, so that  $\mathbf{P}[\mathbf{B} = \mathbf{p}] = 0$  if there is any attached site  $\mathbf{s} \in A$  such that  $\|\mathbf{p} - \mathbf{s}\| > \ell$ .

We call all values of  $\mathbf{p}$  that satisfy  $\|\mathbf{p} - \mathbf{s}\| \leq \ell$  for all  $\mathbf{s} \in A$  the *feasible body positions*, as it is possible the body is in that position when a reaction finally occurs. The set of all feasible positions is denoted  $F$ , and is illustrated as dark blue in Fig. 2.

The exact distribution of  $\mathbf{B}$  at equilibrium will be a Boltzmann distribution over the feasible positions  $\mathbf{p} \in F$  according to the energy  $E(\mathbf{p})$  at each of those positions,

$$\mathbf{P}[\mathbf{B} = \mathbf{p}] = p_B(\mathbf{p}) = \frac{e^{-\beta E(\mathbf{p})}}{\int_F e^{-\beta E(\mathbf{p})} d\mathbf{p}}. \quad (2)$$

In Eq. 2,  $\beta = 1/k_B T$ , where  $k_B$  is Boltzmann's constant and  $T$  is absolute temperature. We are not concerned with temperature variation, so  $T$  will be constant.

## 2.4 Leg-Site Interactions

The kinetics of the bimolecular reaction of leg-site binding is controlled by two factors: the probability of the leg being proximate to a site, and the probability that the leg and

site molecules have enough energy to surmount a reaction energy barrier while they are proximate. These probabilities are controlled by the diffusion of the reactants and the activation energy barrier of the reaction [6]. Depending on which of these two processes is rate-limiting, there are two different types of kinetics for the leg-site binding reactions. If the energy barrier is relatively low, the leg is likely to react with one of the first few sites it comes in contact with. Thus, the leg will be more likely to react with sites closer to where it had previously been attached. Because the diffusion to new sites is the limiting factor in the reaction this situation is called *diffusion-limited*. On the other hand, if the energy barrier is higher, the probability of gaining enough kinetic energy from thermal fluctuations will be the limiting factor. This situation is called *reaction-limited*. The leg will diffuse around until there is enough energy to react, and because the leg is constrained to move over a small area, it will quickly reach an equilibrium distribution over sites.

At present we consider the reaction-limited case. Under these conditions, we can assume that the probability of a leg attaching to a site is proportional to the probability that the body is in a position that is less than distance  $\ell$  from the site. We define a function

$$f_L(\mathbf{p}) = \begin{cases} 1 & \|\mathbf{p}\| \leq \ell \\ 0 & \text{otherwise} \end{cases},$$

that determines if a site at position  $\mathbf{p}$  is feasible from the origin.

Now, because the body has a distribution over feasible positions, some sites can only be reached from a portion of the feasible body positions. For any site  $\mathbf{s}$ , we can define the probability for leg  $i$  being proximate to  $\mathbf{s}$  when it reacts as

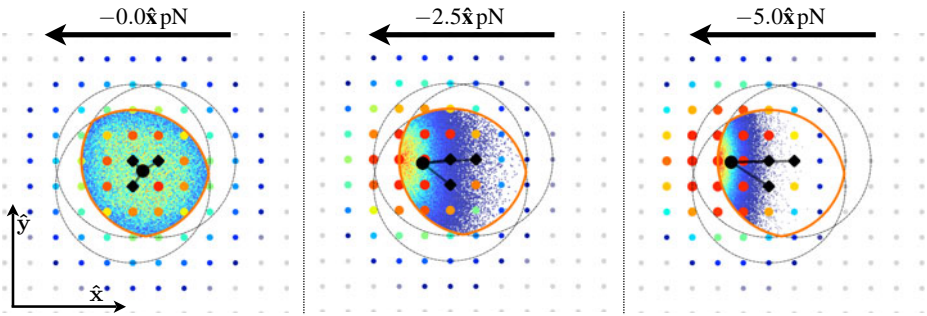
$$\mathbf{P}[i \text{ proximate to } \mathbf{s}] = \int_F p_B(\mathbf{p}) f_L(\mathbf{s} - \mathbf{p}) d\mathbf{p}. \quad (3)$$

Any site with non-zero probability of being reached is called a *feasible site* and this defines the region of feasible sites shown in Fig. 2 in light blue. Sites outside the feasible region have a rate of 0 for attachment to a leg. If a leg is proximate to a site, there is a constant rate per unit time at which the particular leg-site binding will occur. This rate is a function of the diffusion rates, leg structure, and chemical free energy barriers, but as these are constants we ignore the details and just assume that the rate is  $k_+(\pi(\mathbf{s}))$  when the site  $\mathbf{s}$  has species  $\pi(\mathbf{s})$ . Then the rate of attachment for unattached leg  $i$  to feasible site  $\mathbf{s}$  is  $k_+(\pi(\mathbf{s}))\mathbf{P}[i \text{ proximate to } \mathbf{s}]$ . Together with the much simpler rates for the unimolecular dissociation and cleavage reactions, which are independent of body and leg diffusion, this enables us to model all of the reactions that lead to state transitions in the model.

## 2.5 Effect of Forces on Walkers

The MVRW model can also model the effect of forces on the body of the walker. This is an advantage of modeling the body's distribution as a Boltzmann distribution determined by the energy of the spider at each feasible position. Consider the original energy function  $E(\mathbf{p})$ . Under the effect of a force  $\mathbf{f}$ , the new energy of position  $\mathbf{p}$  becomes,

$$\tilde{E}(\mathbf{p}) = E(\mathbf{p}) - \mathbf{f} \cdot (\mathbf{p} - \mathbf{p}_0). \quad (4)$$



**Fig. 3.** The equilibrium body distribution for a walker under several different forces:  $-0.0\hat{x}$  pN,  $-2.5\hat{x}$  pN, and  $-5.0\hat{x}$  pN. As the force increases the energy of positions in the  $+\hat{x}$  direction become higher, and their probability decreases. The body is drawn at the distribution mean, and the color and size of sites indicates their effective rate for attachment reactions

Because the probability is determined by a Boltzmann distribution, the absolute value of the energy doesn't matter, so any  $\mathbf{p}_0$  reference point will do for determining the energy of the positions.

The new energy  $\tilde{E}$  will give a new equilibrium distribution whose probability mass is shifted in the direction of the applied force. The effect of forces on the body's equilibrium position, and the propensity for each of the feasible sites is shown graphically in Fig. 3.

### 3 Simulation

The MVRW model takes the form of a discrete-space, continuous-time Markov process (CTMP). Let  $\Omega$  be the set of states for the MVRW process, and recall that, as described in Section 2.1, each state  $\omega \in \Omega$  can be defined as a triple  $(\pi, d, A)$ . Then, given settings for the relevant model parameters and a suitable start state  $\omega_0 = (\pi_0, d_0, A_0)$ , the MVRW Markov process is described by  $X(t)$ , where for each  $t \in [0, \infty)$ ,  $X(t)$  is a random variable over  $\Omega$  giving the distribution for the state of the process at that time. A full characterization of the Markov process would involve analytic estimates for the probability distributions  $X(t)$ . However, this is both infeasible and unnecessary for our purposes.

#### 3.1 Monte Carlo Simulation

A more tractable way to analyze CTMP's is through the Monte Carlo approach. A *Monte Carlo simulation* generates a function  $x : [0, t_{\max}] \rightarrow \Omega$ , called a *realization* of  $X(t)$ . At each time  $t$ ,  $x(t)$  is a sample of the random variable  $X(t)$ .

Discrete-state Markov processes must jump instantaneously from one state to the next, hence such processes are often called *jump Markov processes* [4]. A jump Markov process can be described by a transition rate function  $Q$ , where  $Q(\omega_1 \rightarrow \omega_2) \geq 0$  gives the rate of transition (jumping) from state  $\omega_1$  to state  $\omega_2$ . This function, and an initial





**Fig. 4.** (a) At step  $n$  of the KMC algorithm, the system is in state  $s_n$ , and we must choose  $s_{n+1}$  from amongst the  $k$  possible next states  $\{z_i\}_{i=1}^k$  according to their respective transition rates  $\{r_i\}_{i=1}^k$ . (b) We can select the next state with a single random number  $\alpha \sim \text{Uniform}((0, R))$ , where  $R = \sum r_i$  is the total rate. This example shows the next state chosen to be  $z_2$ .

start state  $\omega_0$ , completely determine the Markov process  $X(t)$ . For jump Markov processes, Monte Carlo simulation can be carried out exactly, because a realization  $x(t)$  will be a piecewise constant function, consisting of jumps to a sequence of states  $\{s_i\}$  at a sequence of jump times  $\{t_i\}$ , so that  $x(t) = s_i$  for  $t \in [t_i, t_{i+1})$ .

There are two main uses for Monte Carlo simulations of Markov processes. The first is to estimate the equilibrium distribution of Markov processes with a limiting distribution. In this approach the state sequence  $\{s_i\}$  becomes an unbiased sampling from a distribution that would otherwise be hard to sample from. The second use is to estimate the dynamic or kinetic properties of a Markov process as it evolves from its initial state. In this case we are interested in how an out-of-equilibrium Markov process behaves as it evolves according to the transition function  $Q$ .

To study the MVRW model we use both types of Monte Carlo simulations. At the timescales of chemical reactions we use the kinetic Monte Carlo algorithm to simulate the dynamics of the MVRW Markov processes, obtaining traces of individual spiders moving stochastically according to transition rates. In contrast, at the physical timescales we use the Metropolis-Hastings algorithm to sample from the equilibrium distribution of the body's position as it moves by constrained diffusion in the feasible region  $F$ .

### 3.2 The Kinetic Monte Carlo Algorithm

The kinetic Monte Carlo (KMC) method refers to a rejection-free method of generating exact realizations of a jump Markov process by starting at some fixed initial state and evolving the system state and time according to the transition rates of the model [13]. Let  $X(t)$  be a Markov process over state space  $\Omega$  with transition rate function  $Q$ . Given an initial state  $s_0$ , the KMC algorithm evolves the system state through time. After the  $n$ -th step of the algorithm, the system will be in state  $s_n$  at time  $t_n$ . The task of the KMC algorithm is to stochastically choose  $s_{n+1}$  and  $t_{n+1}$  according to the transition rates,  $Q$ . Let  $Z = \{s' \in \Omega \mid Q(s_n \rightarrow s') > 0\}$  be the set of transitions from state  $s_n$  with non-zero rate. We assume that  $|Z| = k$  is finite and non-zero, and thus we can enumerate it as  $Z = \{z_i\}_{i=1}^k$ , and define rates  $\{r_i\}_{i=1}^k$ , with  $r_i = Q(s_n \rightarrow z_i)$ . Let the total rate of all transitions be  $R = \sum_{i=1}^k r_i$ . This situation is illustrated in Figure 4a.

The probability of the process moving to state  $z_i$  at step  $n+1$  is given by the ratio  $r_i/R$ . We can choose a next state  $z^* \in Z$  by selecting a random number  $\alpha \sim \text{Uniform}([0, R])$

and choosing  $z^* = z_j$ , where  $j$  is the smallest integer satisfying  $\sum_{i=1}^j r_i > \alpha$ . This process is depicted in Figure 4b.

Finally, the algorithm decides how much time should elapse until the transition to  $z^*$ . From our current state, all of the possible transitions in  $Z$  occur stochastically with constant rate per unit time. Thus, the time  $\tau_i$  until the transition to  $z_i$  will be exponentially distributed,  $\tau_i \sim \text{Exp}(r_i)$ . We are interested only in the probability distribution for the minimum of these times,  $\tau^* = \min\{\tau_1, \dots, \tau_k\}$ . The exponential distribution has the convenient property that  $\tau^*$  will also be exponentially distributed, as

$$\mathbf{P}[\min\{\tau_1, \dots, \tau_k\} > t] = \mathbf{P}\left[\bigwedge_{i=1}^k \tau_i > t\right] = \prod_{i=1}^k \mathbf{P}[\tau_i > t] = \prod_{i=1}^k e^{-tr_i} = e^{-t\sum r_i} = e^{-tR}.$$

Thus, we see that  $\tau^* \sim \text{Exp}(R)$ . Sampling from the exponential distribution is particularly easy, as  $\tau^* = -\ln \beta / R$  for  $\beta \sim \text{Uniform}((0, 1))$ .

At this point, the KMC algorithm records the next state  $s_{n+1} = z^*$  and the new time  $t_{n+1} = t_n + \tau^*$ , and then the process repeats until  $N$  simulation steps have been made.

### 3.3 Metropolis-Hastings Distributions

The MVRW model assumes that the body and unattached legs come to an equilibrium distribution in between reaction steps. In Section 2.4 we explain how the transition rates for binding reactions are computed given a probability distribution  $p_{\mathbf{B}}(\mathbf{p})$  over body locations, and a probability distribution  $p_L(d)$  for the leg’s distance from the body. Overall, the rate at which unbound leg  $i$  binds to site  $\mathbf{s}$  is given by Eq. 3.

With knowledge of  $p_{\mathbf{B}}$ , we can estimate the rates  $r_{i \rightarrow \mathbf{s}}$  using Monte Carlo integration. If  $\mathbf{P}_1, \dots, \mathbf{P}_n \sim \mathbf{B}$  are samples from  $\mathbf{B}$ , they can be used as an unbiased estimator for a function  $f$  of the body’s position [7],

$$\int_F f(\mathbf{p}) p_{\mathbf{B}}(\mathbf{p}) d\mathbf{p} = \left\langle \frac{1}{n} \sum_{i=1}^n f(\mathbf{P}_i) \right\rangle.$$

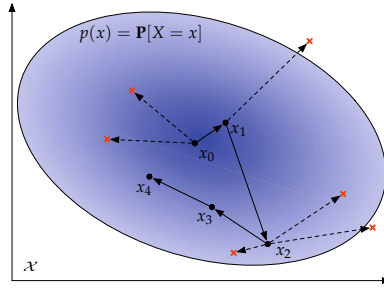
The distribution  $p_{\mathbf{B}}$  is defined in Eq. 2. The denominator in this function,

$$Z = \int_F e^{-\beta E(\mathbf{p})} d\mathbf{p}, \tag{5}$$

is called the partition function, and is difficult to compute making sampling directly from  $p_{\mathbf{B}}$  difficult. The Metropolis-Hastings (MH) algorithm [10, 5] allows  $p_{\mathbf{B}}$  to be sampled without knowledge of  $Z$ .

The MH algorithm samples from  $p_{\mathbf{B}}$  by starting with any Markov process on the distribution domain,  $\mathbb{R}^2$ , transforming that Markov process into an ergodic discrete-time Markov chain that has  $p_{\mathbf{B}}$  as an equilibrium distribution. This Markov chain is defined by transition probabilities  $\tilde{Q}$  where  $\tilde{Q}(\mathbf{p}_1 \rightarrow \mathbf{p}_2) = Q(\mathbf{p}_1 \rightarrow \mathbf{p}_2)\alpha$ , and

$$\alpha = \min \left\{ 1, \frac{p_{\mathbf{B}}(\mathbf{p}_2)Q(\mathbf{p}_2 \rightarrow \mathbf{p}_1)}{p_{\mathbf{B}}(\mathbf{p}_1)Q(\mathbf{p}_1 \rightarrow \mathbf{p}_2)} \right\}. \tag{6}$$



**Fig. 5.** The Metropolis-Hastings algorithm samples from probability distribution  $p(x)$ , by simulating a Markov chain with an equilibrium distribution equal to  $p(x)$ . The algorithm generates a sequence of points  $\{x_i\}_{i=0}^N$  by using the current point to draw a new candidate point, and choosing to accept or reject that point with probability  $\alpha$ . In this figure a red cross represents a rejected point, and a labeled black point represents an accepted point.

The MH algorithm can simulate the Markov process under  $\tilde{Q}$  without ever constructing a rate table explicitly. Also, because the definition of  $\alpha$  has  $p_{\mathbf{B}}$  in the numerator and denominator, the partition function  $Z$  will cancel eliminating the need to compute it. Together these considerations make the MH algorithm an efficient and effective means of sampling from  $p_{\mathbf{B}}$ .

The result of the MH algorithm is a sequence of values  $\{\mathbf{p}_i\}_{i=0}^N$ . At step  $i$ , the simulation has value  $\mathbf{p}_i$  and it uses this to draw a candidate value  $\mathbf{p}^* \sim q(\mathbf{p}) = Q(\mathbf{p}_i \rightarrow \mathbf{p})$ . If Eq. 2 is written as  $p_{\mathbf{B}}(\mathbf{p}) = f(\mathbf{p})/Z$ , then we calculate  $f(\mathbf{p}^*)$  and with Eq. 6 get

$$\alpha = \min \left\{ 1, \frac{f(\mathbf{p}^*)Q(\mathbf{p}^* \rightarrow \mathbf{p}_i)}{f(\mathbf{p}_i)Q(\mathbf{p}_i \rightarrow \mathbf{p}^*)} \right\}.$$

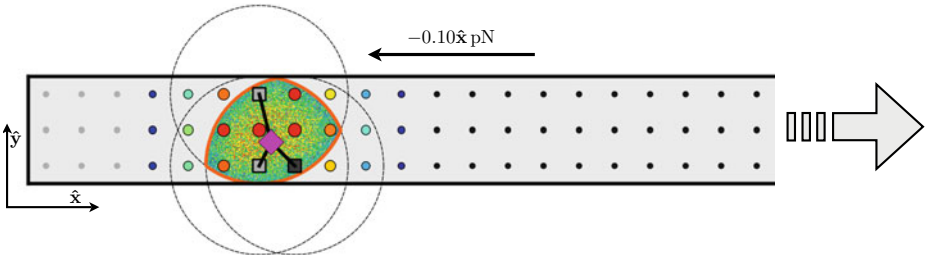
With probability  $\alpha$  we choose to accept the point and set  $\mathbf{p}_{i+1} = \mathbf{p}^*$ , otherwise we reject this candidate value and try again. We repeat this until we have generated  $N$  values. This procedure is illustrated in Figure 5. The sequence of values returned will include an initial period before the chain reaches equilibrium. The initial points are highly dependent on the starting value, and thus are not an unbiased sample. Typically these points are dropped. Depending on the nature of the distribution and the application a threshold can be set so that subsequent points are independent of the starting value with high probability [2].

## 4 Preliminary Results

We used our KMC algorithm to simulate 100 realizations of the MVRW Markov process for several different parameter values. The walkers were simulated until time  $t_{\max} = 3.0 \times 10^6$  s. In our experiment the walker started at the origin on a semi-infinite track. The track is 3 sites wide and sites are on a  $1.0 \text{ nm} \times 1.0 \text{ nm}$  grid, as shown in Fig. 6. The walkers experienced a force in the  $-\hat{x}$  direction that essentially opposed the direction of highest substrate gradient. The simulations parameters are summarized in

**Table 1.** Parameters used in simulations

Parameter	Description
$k = 4$	Number of legs
$\ell = 2.5 \text{ nm}$	Length of each leg
$k_+(S) = k_+(P) = 1.0 \times 10^3 \text{ s}^{-1}$	On rate for leg binding
$k_-(P) = 1.0 \text{ s}^{-1}$	Off rate for products
$k_-(S) = 0.0 \text{ s}^{-1}$	Off rate for substrates
$k_{\text{cat}} \in \{1.0 \text{ s}^{-1}, 0.01 \text{ s}^{-1}\}$	Catalysis rate
$f \in \{0.00 \text{ pN}, 0.05 \text{ pN}, 0.10 \text{ pN}, 0.50 \text{ pN}\}$	Force in the $-\hat{x}$ direction
$T = 300.0 \text{ K}$	Absolute temperature



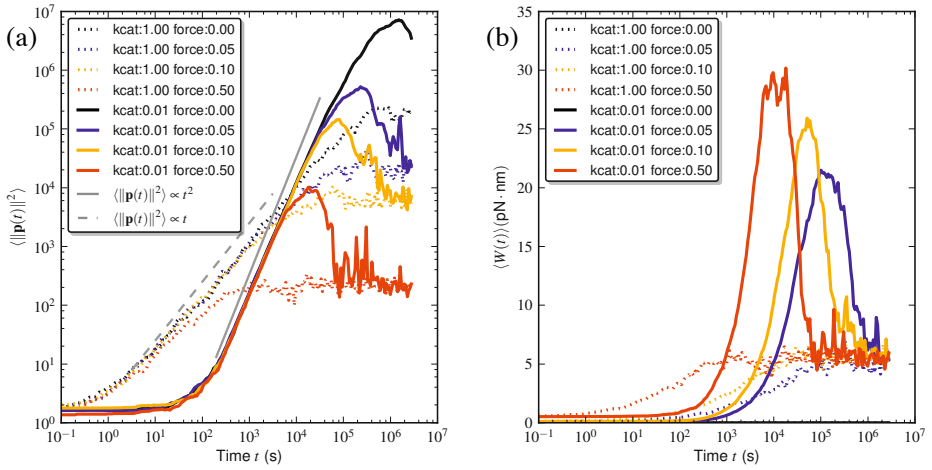
**Fig. 6.** An example configuration of a MVRW simulation. The walker has three attached legs. Light gray sites represent products, dark gray are substrates. Each attached leg forms a circular constraint on the body's position, defining the feasible region in orange. The body's distribution is shown within this feasible region, and the color and size of surrounding sites represents their effective rate for attachment reactions.

Table 1. The only parameters varied were the force and the catalysis rate. The experiments with  $k_{\text{cat}} = 1.0 \text{ s}^{-1}$  have no effective difference between substrate and product, so walkers can only be expected to move diffusively (at least in the absence of any force). However, the walkers with  $k_{\text{cat}} = 0.01 \text{ s}^{-1}$  will experience significantly slower detachment from substrates than from products, producing a residency-time bias.

To quantify the diffusive properties of the walker we estimated moments of several random variables relevant to the walker motion. One of the defining characteristics of diffusive motion is that the mean-squared displacement,  $\langle \|\mathbf{p}\|^2 \rangle$ , of a walker increases as a power law with exponent  $\alpha = 1$ . Eq. 7 defines various forms of anomalous diffusion when  $0 \leq \alpha \leq 2$ , where  $d = 2$  is the dimension and  $D$  is the diffusion constant.

$$\langle \|\mathbf{p}(t)\|^2 \rangle = (2dD)t^\alpha, \quad \begin{cases} \alpha = 0 & \text{stationary} \\ 0 < \alpha < 1 & \text{subdiffusive} \\ \alpha = 1 & \text{diffusive} \\ 1 < \alpha < 2 & \text{superdiffusive} \\ \alpha = 2 & \text{ballistic or linear} \end{cases} \quad (7)$$

In Fig. 7(a) we show the mean-squared displacement of the walkers on a log-log plot where power laws are straight lines. We show reference lines for the power laws



**Fig. 7.** (a) The mean squared displacement of walkers shows significant superdiffusion in walkers with small  $k_{cat}$ . (b) The work done by walkers against an opposing force. The walkers moving under zero force always do zero work. Note that  $\langle W(t) \rangle \rightarrow 4.14 \text{ pN} \cdot \text{nm} = k_B T$ , as  $t \rightarrow \infty$ .

corresponding to diffusive and ballistic motion. Clearly, the walkers with lower  $k_{cat}$  experience significant periods of superdiffusive motion, while the walkers with  $k_{cat} = 1.0 \text{ s}^{-1}$  move mainly diffusively.

If we consider the work done against the opposing force, there is a significant amount of work done on average for all of the walkers, but the maximum mean work is done by walkers with lower  $k_{cat}$ . In Sec. 5 we show that the lower values of  $k_{cat}$  act to bias the walker in the direction of uncleaved substrate, and this direction is essentially in opposition to the force exerted on the walkers, allowing them to use this bias to do work against the force. Eventually, however, all of the walkers move backwards into regions of product sites, and end up effectively diffusing like the walkers with  $k_{cat} = 1.0 \text{ s}^{-1}$  (Fig 7(b)).

## 5 Mechanism of Superdiffusive Motion

The results of Sec. 4 show that spiders can move superdiffusively in the direction of new sites even in opposition to a force. Over significant spans of time, the walkers will have effectively done work against the force as their motion is biased by the chemical energy in the sites they cleave.

Molecular spiders operate by cleaving a substrate oligonucleotide, leaving behind a shorter oligonucleotide product – an irreversible reaction. A molecular spider starting on a substrate-covered surface is a system far from equilibrium, and consequently has the potential to do useful work as it relaxes towards equilibrium. Under the parameters of Table 1, there is a *residency-time bias* between leg-substrate and leg-product bindings for the walkers with  $k_{cat} < 1.0 \text{ s}^{-1}$ , because the leg-substrate bindings are much longer lived than the leg-product bindings. When combined with a non-uniform local

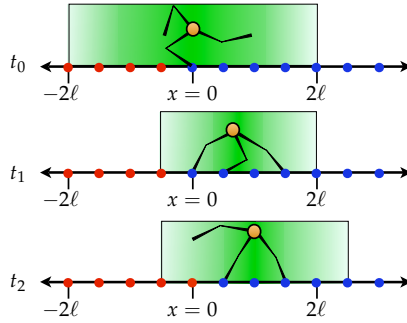
distribution of substrates, the slower unbinding from substrates causes the walker to be effectively biased in the direction higher substrate density.

When a leg binds to a site, it forms a constraint on the position of the body and the actions of the other legs until a dissociation reaction occurs. According to Eq. 1, the rate of detachment for a leg-substrate complex is  $k_{\text{cat}}(\text{S}) + k_{-}(\text{S})$ , versus  $k_{-}(\text{P})$  for a leg-product complex. We define  $r = (k_{\text{cat}}(\text{S}) + k_{-}(\text{S}))/k_{-}(\text{P})$ . If  $r = 1$ , there is effectively no difference between substrate and product; although the substrate sites are transformed to products, they do not affect the behavior of the walker. This is equivalent to a walker moving over an all-product surface – an equilibrium process. Thus, we can expect the walker to undergo normal diffusion when  $r = 1$ . Indeed, this is what we see in Fig. 7a, where the spiders with  $k_{\text{cat}} = 1.0 \text{ s}^{-1}$  move diffusively with  $\langle \|\mathbf{p}(t)\|^2 \rangle \propto t$ .

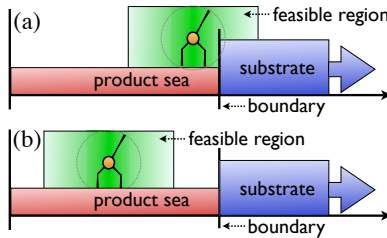
However, when  $0 < r < 1$ , a leg-substrate bond lasts longer than a leg-product bond, and substrates effectively act like anchors. A leg attached to a substrate restricts the movement of the walker body and other legs until the substrate is cleaved, and the other legs are constrained to attach to feasible sites close to the attached leg. If a free leg attaches to a product, it will quickly detach and be free to attach again to another site. If there are any other substrates in the local environment, one of the other free legs will eventually find and attach to one. Thus, the legs are in some sense attracted to substrates, but not because they specifically seek out the substrates or prefer them to products. Instead, the bias is more subtle, caused by a combination of the residency-time bias and the collective constraints on the legs imposed by the connection to a common body. The legs eventually find the substrates simply because if they attach to a product, they will quickly end up detaching and randomly choosing a new attachment site again and again until they find a substrate. Note that this effect is only present when the walker has more than one leg and has  $r < 1$ , so both of these properties are critical for spiders to move superdiffusively.

This bias, however, also depends on the local availability of substrates. Once a leg attaches to a substrate, the site will eventually be irreversibly transformed into a product. Thus, while the legs (passively) seek out the substrates, they eventually will deplete the local substrate supply. For a small environment with a limited number of sites, substrates will all quickly be turned into products, at which point the system will be at equilibrium and the walker will move diffusively. However, with larger environments this march towards equilibrium takes a significant amount of time, and during this non-equilibrium period there is potential for superdiffusive motion and for doing physical work against a force.

Now, consider what happens when the local environment has a non-uniform distribution of substrates. Suppose, as in Fig. 8, the walker has a single leg attached to a substrate at site  $s$  with location  $x = 0$ . The local environment of feasible sites will then consist of all sites within two leg lengths ( $2\ell$ ) from  $x = 0$ . Suppose that all sites with position  $x \geq 0$  are substrates and all sites with position  $x < 0$  are products. Now consider what happens when the process is started. The initially attached leg will likely remain attached to the substrate for some time if  $r < 1$ . During this time the other  $k - 1$  legs will be restricted to the feasible sites. Short lived product attachments mean that legs will end up preferentially attached to substrates by the time the first leg cleaves and detaches. At this point if most of the legs are on substrates, and all of the substrates are



**Fig. 8.** A residency-time bias combined with a non-uniform local distribution of substrates can lead to a directional bias. There is a boundary at  $x = 0$  between substrates (blue) and products (red). At time  $t_0$  a single leg is attached to a substrate, and the other legs can attach to any feasible sites (shaded area). Because the leg-product pairs are short-lived, the legs are more likely to end up attached to substrates at time  $t_1$ . When the first leg detaches at time  $t_2$ , the equilibrium position and substrate boundary will move right.



**Fig. 9.** (a) The walker in a boundary state  $B$  where it is attached to substrates on the boundary between visited and unvisited sites. The residency-time bias and non-uniform local distribution of substrates gives the spider an outward bias. (b) The walker in the diffusive state  $D$  where it moves over previously visited sites.

to the right, the spider’s equilibrium body position will move right. At the same time, because the site at  $x = 0$  is now a product, the boundary between the substrates and products also moves right. Thus, the walker is biased towards moving right, and simultaneously shifts the biasing-inducing substrate/product boundary rightward as well. As long as the walker stays attached to substrates by the boundary, it will tend to move along with the boundary, causing the walker to move ballistically in the direction of new substrates. However, there is still some probability that the walker detaches from all substrates and moves backwards over previously visited sites. In this case, the walker must move diffusively.

In previous work [12] we also observed significant periods of superdiffusive motion in the simpler one-dimensional molecular spider models of Antal and Krapivsky [1]. For these models, we explained this superdiffusive motion by showing that the Markov process can be viewed as consisting of two metastates: a boundary ( $B$ ) state where the walker is on the boundary between cleaved and uncleaved sites, and a diffusive ( $D$ ) state where the walker is moving over previously visited sites. The walker moves

ballistically in the  $B$  state and diffusively in the  $D$  state, and the overall motion depends on how much time the walker spends in each of the metastates. Similarly, the initial superdiffusive motion in the MVRW model for  $r < 1$  can be understood as the walker moving between a  $B$  and a  $D$  state as shown in Fig 9. The walker initially spends most of its time in the  $B$  state, moving ballistically away from the origin in the direction of unvisited sites, and in opposition to the force. However, the walker has a constant probability of falling off the boundary and into the  $D$  state where it moves diffusively over previously visited sites. In the  $D$  state, the force acts to bias the motion of the walker backwards, and as the size of the region of cleaved products (the *product sea*) grows, the spider takes increasingly long to return to the  $B$  state, and eventually becomes on average stationary at some equilibrium position with mean work  $\langle W(t) \rangle = k_B T$ , as observed in Fig 7a.

## 6 Discussion

Given the many potential nanoscale applications for molecular spiders, it is interesting to see that the MVRW model predicts that walkers move superdiffusively over significant times and distances, even in the presence of a force. This motion is not a product of differing  $k_+$  rates, but is rather of a more subtle nature, emerging from the interaction of a residency-time bias, local substrate anisotropy, and constraints imposed by multiple legs attached to a single body. Walkers with  $r < 1$  stay attached to substrate sites longer than to product sites. The presence of a non-uniform local distribution of substrates combined with the constrained diffusion imposed by the attached legs causes the walker to move in the direction of highest substrate density, leading to superdiffusive behavior when  $r < 1$ . This effect relies on the walker having multiple legs.

The ability of the MVRW model to stochastically incorporate the effect of force on the walker kinetics is due to the separation of time scales between the very fast physical motion and vibration of molecules and the much slower chemical reactions. Because the body and unattached legs come to an equilibrium before reattaching, we can model their motion together with the effect of a force using a Boltzmann distribution. This assumption means that the body and unattached leg positions need not be part of the MVRW model state, so our model remains discrete and can be simulated exactly.

Because of our choice to explicitly separate the timescales of the physical and chemical events, the MVRW model uses Monte Carlo simulation separately for both equilibrium and kinetic analysis of Markov processes. The MVRW model is a non-ergodic Markov process describing a system significantly out of equilibrium, and we use kinetic Monte Carlo techniques to observe the simulated stochastic evolution of a walker moving over the sites. This puts us in the position of a virtual experimenter, able to run simulated traces of the spider's motion and measure exactly any desired properties of their motion. In contrast to this kinetic simulation, we use the Metropolis-Hastings algorithm to study the equilibrium distribution of the walker's body moving under the constrained diffusion as enforced by the attached legs.

The superdiffusive motion of walkers in the MVRW model can be understood through the decomposition of the process into a  $B$  metastate where the walker is on the boundary between substrates and products and is moving ballistically, and a  $D$  metastate where



the walker is moving diffusively over product sites. From a practical standpoint, the duration of the superdiffusive effect and the magnitude of the work done against a force can be increased by designing walkers that are less likely to move from the *B* to *D* states. Future work will focus on the how the geometry of the walkers and their kinetic properties can be optimized to increase the amount of time they spend in the *B* state moving ballistically, hence maximizing their utility for faster than diffusion molecular transport and communication.

**Acknowledgments.** This material is based upon work supported by the National Science Foundation under grants 0533065 and 0829896.

## References

1. Antal, T., Krapivsky, P.L.: Molecular spiders with memory. *Physical Review E* 76(2), 21121 (2007)
2. Geyer, C.J.: Practical Markov chain Monte Carlo. *Statistical Science* 7(4), 473–483 (1992)
3. Gillespie, D.T.: A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational Physics* 22, 403–434 (1976)
4. Gillespie, D.T.: *Markov processes*. Academic Press Inc., Boston (1992)
5. Hastings, W.K.: Monte Carlo sampling methods using Markov chains and their applications. *Biometrika* 57(1), 97–109 (1970)
6. Henriksen, N.E., Hansen, F.Y.: *Theories of Molecular Reaction Dynamics*. Oxford University Press, New York (2008)
7. Kalos, M.H., Whitlock, P.A.: *Monte Carlo Methods*. John Wiley & Sons, New York (1986)
8. Lund, K., Manzo, A.J., Dabby, N., Michelotti, N., Johnson-Buck, A., Nangreave, J., Taylor, S., Pei, R., Stojanovic, M.N., Walter, N.G., Winfree, E., Yan, H.: Molecular robots guided by prescriptive landscapes. *Nature* 465, 206–210 (2010)
9. McQuarrie, D.A.: Stochastic approach to chemical kinetics. *Journal of Applied Probability* 4(3), 413–478 (1967)
10. Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H., Teller, E.: Equation of state calculations by fast computing machines. *The Journal of Chemical Physics* 21(6), 1087–1092 (1953)
11. Pei, R., Taylor, S.K., Stefanovic, D., Rudchenko, S., Mitchell, T.E., Stojanovic, M.N.: Behavior of polycatalytic assemblies in a substrate-displaying matrix. *Journal of the American Chemical Society* 128, 12693–12699 (2006)
12. Semenov, O., Olah, M.J., Stefanovic, D.: Mechanism of diffusive transport in molecular spider models. *Physical Review E* 83(2), 021117 (2011)
13. Voter, A.: Introduction to the kinetic Monte Carlo method. In: Sickafus, K., Kotomin, E., Uberuaga, B. (eds.) *Radiation Effects in Solids*. Springer, Heidelberg (2007)

# Exact Shapes and Turing Universality at Temperature 1 with a Single Negative Glue

Matthew J. Patitz<sup>1,\*</sup>, Robert T. Schweller<sup>1,\*</sup>, and Scott M. Summers<sup>2</sup>

<sup>1</sup> Department of Computer Science, University of Texas–Pan American,  
Edinburg, TX, 78539, USA

{mpatitz,schweller}@cs.panam.edu

<sup>2</sup> Department of Computer Science and Software Engineering,  
University of Wisconsin–Platteville, Platteville, WI 53818, USA

summerss@uwplatt.edu

**Abstract.** Is Winfree’s abstract Tile Assembly Model (aTAM) “powerful?” Well, if certain tiles are required to “cooperate” in order to be able to bind to a growing tile assembly (a.k.a., temperature 2 self-assembly), then Turing universal computation and the efficient self-assembly of  $N \times N$  squares is achievable in the aTAM (Rotemund and Winfree, STOC 2000). So yes, in a computational sense, the aTAM is quite powerful! However, if one completely removes this cooperativity condition (a.k.a., temperature 1 self-assembly), then the computational “power” of the aTAM (i.e., its ability to support Turing universal computation and the efficient self-assembly of  $N \times N$  squares) becomes unknown. On the plus side, the aTAM, at temperature 1, is not only Turing universal but also supports the efficient self-assembly  $N \times N$  squares if self-assembly is allowed to utilize three spatial dimensions (Fu, Schweller and Cook, SODA 2011). In this paper, we investigate the theoretical “power” of a seemingly simple, restrictive variant of Winfree’s aTAM in which (1) the absolute value of every glue strength is 1, (2) there is a single negative strength glue type and (3) unequal glues cannot interact (i.e., glue functions must be “diagonal”). We call this abstract model of self-assembly the *restricted glue* Tile Assembly Model (rgTAM). We achieve two positive results. First, we show that the tile complexity of uniquely producing an  $N \times N$  square in the rgTAM is  $O(\log N)$ . In our second result, we prove that the rgTAM is Turing universal.

## 1 Introduction

Even in an overly-simplified model such as Winfree’s abstract Tile Assembly Model (aTAM) [24], the theoretical power of algorithmic self-assembly is formidable. Universal computation is achievable [24] and computable shapes self-assemble as efficiently as the limits of algorithmic information theory will allow [23, 21, 1]. However, these theoretical results all depend on an important

---

\* This author’s research was supported in part by National Science Foundation Grant CCF-1117672.

system parameter, the temperature  $\tau$ , which specifies the minimum amount of binding force that a tile must experience in order to permanently bind to an assembly. The temperature  $\tau$  is typically set to a value of 2 because at this temperature (and above), the mechanism of “cooperation” is available, in which the correct positioning of multiple tiles is necessary before certain additional tiles can attach. However, in temperature 1 systems, where such cooperation is unenforceable, despite the fact that they have been extensively explored [10, 5], it remains an unproven conjecture that self-assembly at temperature  $\tau < 2$  is incapable of universal computation. It is also widely conjectured (most notably in [21]), although similarly unproven, that the efficient self-assembly of such shapes even as simple as  $N \times N$  squares is impossible.

Given the seeming theoretical weakness of tile assembly at temperature 1, contrasted with its computational expressiveness at temperature 2, it seems natural that experimentalists would focus their efforts on the latter. However, as is often the case, what seems promising in theory is not necessarily as promising in practice. It turns out that in laboratory implementations of tile assembly systems [22, 3, 4], it has proven difficult to build true strength-2 glues in addition to being able to strictly enforce the temperature threshold (e.g. many errors that are due to “insufficient attachment” tend to occur in practice). Therefore, the characterization of self-assembly at temperature 1 is of the utmost importance.

With the goal in mind of specifying a model of self-assembly that is closer to the intersection of theoretical power and experimental plausibility, in this paper, we propose “the aTAM at temperature  $\tau = 1 + \epsilon$ ”. We introduce the *restricted glue* Tile Assembly Model (rgTAM), which requires that (1) all glues have strength  $-1, 0$ , or  $1$ , (2) that there is **only one** glue type that exhibits  $-1$  strength (i.e., a repulsive force equivalent in magnitude to the binding force of a strength 1 glue), and (3) the glue function is *diagonal*, which means that a glue of one type interacts only with other glues of the same type. Our goal in developing the rgTAM is to study the “simplest” model of algorithmic self-assembly that retains the computational and geometrical expressiveness of temperature 2 self-assembly. In this paper, we achieve two positive results. First, we show that the tile complexity of uniquely producing an  $N \times N$  square in the rgTAM is  $O(\log N)$ . In our second result, we prove that the rgTAM is Turing universal.

The use of glues possessing negative strength values has been investigated within a variety of contexts [18, 7]. However, previous results have been much less restrictive, allowing non-diagonal glue functions (meaning that glue types can have interactions, perhaps of different strengths, with multiple different glue types) and large magnitudes. Additionally, no explicit bound has been set on the number of unique negative strength glue types. In order to help bridge the gap between theory and experiment, we have proposed restrictions on the aTAM (in the form of the rgTAM as stated above).

Various experimental implementations of the Tile Assembly Model have utilized tiles created from DNA [22, 3, 4, 25, 17]. Moreover, several results have shown that magnetic particles can be attached to DNA molecules [13, 19]. Since two magnetized particles of the same polarity experience a repulsive force, the com-

combination of DNA tiles with attached magnetic particles is a natural prospect for the implementation of negative strength glues. It should be possible to adjust the size, composition, and position of the magnetic particles to cause the repulsive force experienced between two tiles to be roughly equal in magnitude to the attractive force experienced by two strength 1 glues. (Note that utilizing magnetic polarity for glues has previously been modeled in [16].) Also, the requirement of only a single negative glue type allows for the attachment of the same magnetized particle to any tile surface that needs to exhibit a  $-1$  strength glue.

The organization of this paper is as follows. In Section 2, we review the aTAM and define the rgTAM, along with a few other definitions used in our subsequent constructions. In Section 3, we prove that  $N \times N$  squares efficiently self-assemble in the rgTAM. In Section 4, we show how to simulate zig-zag systems (e.g., a tile set that simulates a Turing machine on some input) in the rgTAM.

## 2 Preliminaries

In this paper, we work in the 2-dimensional discrete Euclidean space  $\mathbb{Z}^2$ .

Let  $U_2 = \{(0, 1), (1, 0), (0, -1), (-1, 0)\}$  be the set of all *unit vectors*, i.e., vectors of length 1 in  $\mathbb{Z}^2$ . We write  $[X]^2$  for the set of all 2-element subsets of a set  $X$ . All *graphs* here are undirected graphs, i.e., ordered pairs  $G = (V, E)$ , where  $V$  is the set of *vertices* and  $E \subseteq [V]^2$  is the set of *edges*. All logarithms are base-2.

### 2.1 The Abstract Tile Assembly Model

We now give a brief and intuitive sketch of the Tile Assembly Model that is adequate for reading this paper. More formal details and discussion may be found in [24, 21, 20, 14].

Intuitively, a tile type  $t$  is a unit square that can be translated, but not rotated, having a well-defined “side  $\mathbf{u}$ ” for each  $\mathbf{u} \in U_2$ . Each side  $\mathbf{u}$  of  $t$  has a “glue” of “color”  $\text{col}_t(\mathbf{u})$  – a string over some fixed alphabet  $\Sigma$  – and “strength”  $\text{str}_t(\mathbf{u})$  – an integer – specified by its type  $t$ . Two tiles  $t$  and  $t'$  that are placed at the points  $\mathbf{a}$  and  $\mathbf{a} + \mathbf{u}$ , respectively, interact with *strength*  $\text{str}_t(\mathbf{u})$  if and only if  $(\text{col}_t(\mathbf{u}), \text{str}_t(\mathbf{u})) = (\text{col}_{t'}(-\mathbf{u}), \text{str}_{t'}(-\mathbf{u}))$ . If  $\text{str}_t(\mathbf{u}) > 0$ , those tiles *bind* with that strength.

Given a set  $T$  of tile types, an *assembly* is a partial function  $\alpha : \mathbb{Z}^2 \dashrightarrow T$ , with points  $\mathbf{x} \in \mathbb{Z}^2$  at which  $\alpha(\mathbf{x})$  is undefined interpreted to be empty space, so that  $\text{dom } \alpha$  is the set of points with tiles.  $\alpha$  is *finite* if  $|\text{dom } \alpha|$  is finite. For assemblies  $\alpha$  and  $\alpha'$ , we say that  $\alpha$  is a *subconfiguration* of  $\alpha'$ , and write  $\alpha \sqsubseteq \alpha'$ , if  $\text{dom } \alpha \subseteq \text{dom } \alpha'$  and  $\alpha(\mathbf{x}) = \alpha'(\mathbf{x})$  for all  $\mathbf{x} \in \text{dom } \alpha$ .

A *grid graph* is a graph  $G = (V, E)$  in which  $V \subseteq \mathbb{Z}^2$  and every edge  $\{\mathbf{a}, \mathbf{b}\} \in E$  has the property that  $\mathbf{a} - \mathbf{b} \in U_2$ . The *binding graph* of an assembly  $\alpha$  is the grid graph  $G_\alpha = (V, E)$ , where  $V = \text{dom } \alpha$ , and  $\{\mathbf{m}, \mathbf{n}\} \in E$  if and only if (1)  $\mathbf{m} - \mathbf{n} \in U_2$ , (2)  $\text{col}_{\alpha(\mathbf{m})}(\mathbf{n} - \mathbf{m}) = \text{col}_{\alpha(\mathbf{n})}(\mathbf{m} - \mathbf{n})$ , (3)  $\text{str}_{\alpha(\mathbf{m})}(\mathbf{n} - \mathbf{m}) =$

$\text{str}_{\alpha(\mathbf{m})}(\mathbf{m} - \mathbf{n})$ , and (4)  $\text{str}_{\alpha(\mathbf{m})}(\mathbf{n} - \mathbf{m}) > 0$ . An assembly is  $\tau$ -stable, where  $\tau \in \mathbb{N}$ , if it cannot be broken up into smaller assemblies without breaking bonds of total strength at least  $\tau$  (i.e., if every cut of  $G_\alpha$  cuts edges the sum of whose strengths is at least  $\tau$ ). For the case of negative strength glues, we employ the model of irreversible assembly as defined in [7].

Self-assembly begins with a *seed assembly*  $\sigma$  (typically assumed to be finite and  $\tau$ -stable) and proceeds asynchronously and nondeterministically, with tiles adsorbing one at a time to the existing assembly in any manner that preserves stability at all times.

A *tile assembly system (TAS)* is an ordered triple  $\mathcal{T} = (T, \sigma, \tau)$ , where  $T$  is a finite set of tile types,  $\sigma$  is a seed assembly with finite domain, and  $\tau$  is the temperature. In subsequent sections of this paper, unless explicitly stated otherwise, we assume that  $\tau = 1$  and  $\sigma$  consists of a single seed tile type placed at the origin. An *assembly sequence* in a TAS  $\mathcal{T} = (T, \sigma, 1)$  is a (possibly infinite) sequence  $\alpha = (\alpha_i \mid 0 \leq i < k)$  of assemblies in which  $\alpha_0 = \sigma$  and each  $\alpha_{i+1}$  is obtained from  $\alpha_i$  by the “ $\tau$ -stable” addition of a single tile, where “ $\tau$ -stable” applies to the entire assembly. The *result* of an assembly sequence  $\alpha$  is the unique assembly  $\text{res}(\alpha)$  satisfying  $\text{dom } \text{res}(\alpha) = \bigcup_{0 \leq i < k} \text{dom } \alpha_i$  and, for each  $0 \leq i < k$ ,  $\alpha_i \sqsubseteq \text{res}(\alpha)$ . If  $\alpha = (\alpha_i \mid 0 \leq i < k)$  is an assembly sequence in  $\mathcal{T}$  and  $\mathbf{m} \in \mathbb{Z}^2$ , then the  $\alpha$ -index of  $\mathbf{m}$  is  $i_\alpha(\mathbf{m}) = \min\{i \in \mathbb{N} \mid \mathbf{m} \in \text{dom } \alpha_i\}$ . That is, the  $\alpha$ -index of  $\mathbf{m}$  is the time at which any tile is first placed at location  $\mathbf{m}$  by  $\alpha$ . For each location  $\mathbf{m} \in \bigcup_{0 \leq i < l} \text{dom } \alpha_i$ , define the set of its input sides  $\text{IN}^\alpha(\mathbf{m}) = \{\mathbf{u} \in U_2 \mid \mathbf{m} + \mathbf{u} \in \text{dom } \alpha_{i_\alpha} \text{ and } \text{str}_{\alpha_{i_\alpha}(\mathbf{m})}(\mathbf{u}) > 0\}$ .

We write  $\mathcal{A}[\mathcal{T}]$  for the *set of all producible assemblies of  $\mathcal{T}$* . An assembly  $\alpha$  is *terminal*, and we write  $\alpha \in \mathcal{A}_\square[\mathcal{T}]$ , if no tile can be stably added to it. A TAS  $\mathcal{T}$  is *directed*, or *produces a unique assembly*, if it has exactly one terminal assembly i.e.,  $|\mathcal{A}_\square[\mathcal{T}]| = 1$ . A set  $X$  *strictly self-assembles* if there is a TAS  $\mathcal{T}$  for which every assembly  $\alpha \in \mathcal{A}_\square[\mathcal{T}]$  satisfies  $\text{dom } \alpha = X$ .

## 2.2 Restricted Glue and Zig-Zag Tile Assembly Systems and Path Simulation

**Restricted Glue Tile Assembly Systems.** We say that a tile set  $T$  is *glue restricted* if (1) the absolute value of every glue strength in  $T$  is at most 1, and (2) there is a single negative-strength glue type. Intuitively, glue restricted tile sets are as “close” as one can get to *pure* temperature one self-assembly in the aTAM. Finally, a TAS  $\mathcal{T} = (T, \sigma, 1)$  is *glue restricted* if  $T$  is glue restricted. In this paper, for notational convenience, we will simply refer to a glue restricted TAS as a restricted TAS, and the model as the *restricted glue* Tile Assembly Model, or rgTAM.

**Zig-Zag Tile Assembly Systems.** A tile assembly system  $\mathcal{T} = (T, \sigma, 2)$  is called a *zig-zag* [5] tile assembly system if (1)  $\mathcal{T}$  is directed, (2) there is a unique assembly sequence  $\alpha$  in  $\mathcal{T}$ , with  $\mathcal{A}_\square[\mathcal{T}] = \{\alpha = \text{res}(\alpha)\}$ , (3) for every  $\mathbf{x} \in \text{dom } \alpha$ ,  $(0, 1) \notin \text{IN}^\alpha(\mathbf{x})$  and (4) for every  $\mathbf{x} \in \text{dom } \alpha$  and every  $\mathbf{u} \in U_2$ ,  $\text{str}_{\alpha(\mathbf{x})}(\mathbf{u}) + \text{str}_{\alpha(\mathbf{x})}(-\mathbf{u}) < 4$ . Intuitively, zig-zag systems are those that grow exactly one

horizontal row at a time with successive rows being initiated at “opposite” ends of the growing assembly. Zig-zag systems always add new rows to the north (never to the south) and do not grow via a sequence of consecutive double bonds. Zig-zag systems are capable of simulating universal Turing machines, and thus universal computation [5].

**Path Simulation.** Let  $\mathcal{T} = (T, \sigma, 2)$  be a zig-zag TAS with assembly sequence  $\alpha = (\alpha_i \mid 0 \leq i < k)$ . We say that a restricted TAS  $\mathcal{S} = (S, \sigma', 1)$  *path simulates*  $\mathcal{T}$  (at *scale factor*  $c$ ) if  $\mathcal{S}$  uniquely produces a single-tile-wide path of tiles that is logically divided into segments of some fixed-length  $c \in \mathbb{N}$ , where each such sub-path of tiles corresponds to exactly one tile in  $T$ , and these sub-paths self-assemble exactly in accordance with the unique assembly sequence of  $\mathcal{T}$ . In more formal terms,  $\mathcal{S}$  path simulates  $\mathcal{T}$  at scale factor  $c$  if there exists an assembly sequence  $\beta = (\beta_j \mid 0 \leq j < l)$  in  $\mathcal{S}$  such that, for all  $0 \leq i < k$ , the assembly subsequence  $\beta_{ic}, \dots, \beta_{(i+1)c-1}$  in  $\mathcal{S}$  corresponds to the single-tile extension that yields  $\alpha_i$  and the binding graph of  $\beta = \text{res}(\beta)$  is a path. Note that the idea of one tile assembly system simulating another has been studied in other contexts as well [8, 5, 15].

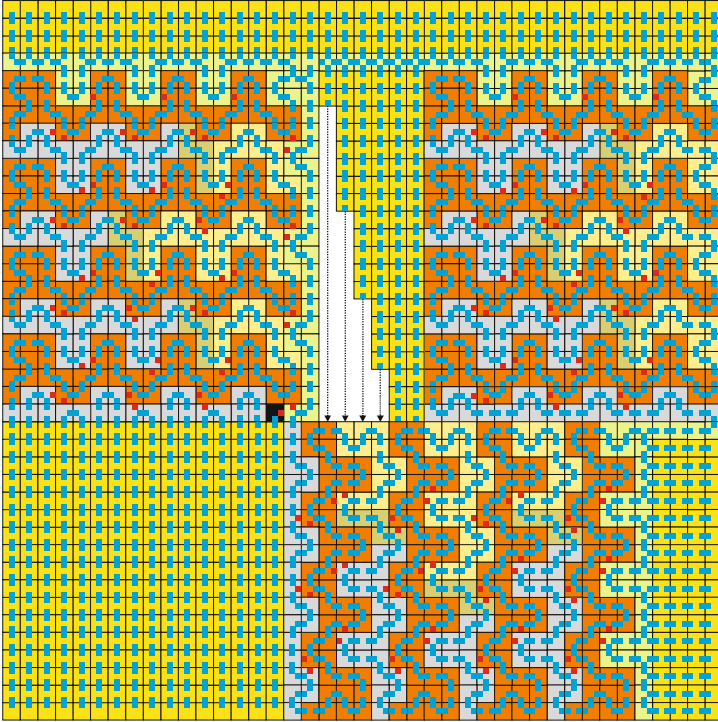
### 3 Exact Shapes

The self-assembly of  $N \times N$  squares has been studied extensively (see [1, 6, 21, 11, 9, 12]). Rothemund and Winfree conjectured in [21] that, for every  $N \in \mathbb{N}$ , if  $\mathcal{T}_N = (T_N, \sigma, 1)$  uniquely produces  $S_N = \{0, 1, \dots, N - 1\} \times \{0, 1, \dots, N - 1\}$ , then  $|T_N| \geq 2N - 1$ . In what follows, we show that this bound does not hold for restricted tile assembly systems.

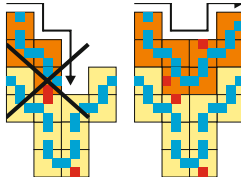
**Theorem 1.** *For all but finitely many  $N \in \mathbb{N}$ , there exists a restricted TAS  $\mathcal{T} = (T_N, \sigma, 1)$ , such that  $S_N$  strictly self-assembles in  $\mathcal{T}$ ,  $\mathcal{T}$  is directed, and  $|T_N| = O(\log N)$ .*

In what follows, we briefly sketch our construction for Theorem 1. Let  $n = \lfloor \frac{N-1}{5} \rfloor$ ,  $k = \lceil \log n \rceil$ ,  $K = 5 + 4k$ ,  $n_0 = 2^k - n + \lceil \frac{K}{5} \rceil$  and  $x = N - (K + 5(n - \lceil \frac{K}{5} \rceil))$ . Intuitively,  $N$  is the dimension (length of one side) of the target square  $S_N$ ,  $n$  is the number of count/increment row pairs that we will need in our construction,  $k$  is the *logical* width of the counter,  $K$  is the *actual* width of the counter in our construction,  $n_0$  is the initial value for the counter,  $2^k - 1$  is the maximum value of the counter and  $x$  is the number of rows on top of the counter that we need to fill in with generic “filler” tiles. Although not necessarily surprising, it is worthy of note—and easy to show—that  $x \leq 9$ . In Figure 1, we show a high-level overview of the terminal assembly produced by our construction (many details are omitted).

In our construction, we emulate the zig-zag counter of Rothemund and Winfree [21]. We utilize three different binary counters in our construction, denoted as the *first*, *second* and *third* counter and oriented vertically, horizontally and vertically respectively. We will discuss the general behavior of our north-growing zig-zag counter and highlight any subtle differences between the two other versions of it that we use in our construction.



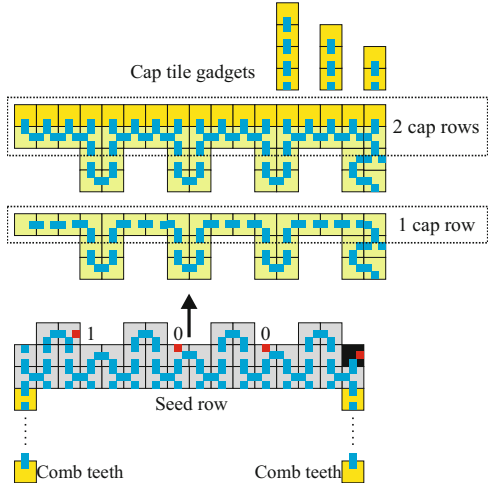
**Fig. 1.** The negative glue is denoted as a little red square. Positive glues are denoted as little blue squares. Yellow represents filler tiles, grey either seed row tiles or “no carry” tiles, orange represents copy tiles, light yellow represents “search for rightmost 0” tiles, dark yellow represents “flip rightmost 0 to 1” tiles and green tiles represent logical connections between the three different counters in our construction. Our construction is simple: it merely assembles a ‘U’ shape via three counters and then fills in the “interior” of the ‘U’ via generic filler tiles. In this example,  $N = 41$ ,  $n = 8$ ,  $k = 3$ ,  $K = 17$ ,  $n_0 = 4$  and  $x = 4$ .



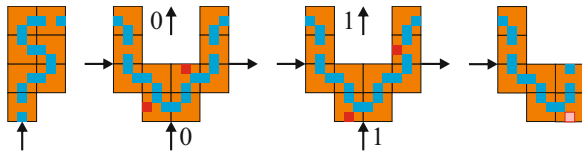
**Fig. 2.** We emulate “temperature-2” style cooperation by using geometry along with the careful placement of the negative glue in order to ensure that only the “correct” tile in a particular location of a path of tiles attaches and therefore can “know” if it is supposed to be, for example, a ‘1’ or a ‘0’ bit. We use this technique extensively throughout this paper.

The binary counter consists of a seed row, which encodes some number in binary, on top of which some number of increment/copy row pairs self-assemble in a zig-zag fashion. The top of the counter is capped off with a special cap gadget.

**The Seed Row.** The seed row is a row of tiles that encodes the initial value of the binary counter  $n_0$  using  $k = \lceil \log n \rceil$  bits and has a horizontal extent of  $K - 1$ . The counter starts counting at this value and stops at  $2^k - 1$ . We encode the bits of  $n_0$  via the careful placement of the negative glue (denoted as a little red square in all of our figures). The bit 0 is encoded by positioning the negative glue so that it is facing north in a dent and a 1 is encoded by positioning the negative glue so that it is facing east in a dent; see Figure 3. This bit encoding scheme is also used in count rows whereas slightly different encoding is used for copy rows. Off the bottom of the seed row, teeth of a “comb” attach in order to fill in the bottom left corner of the square. Each tooth has length  $K$  and self-assembles to the south. The actual length of the seed row—and hence the actual width of our counter in this construction—is  $K$ .



**Fig. 3.** The grey tiles encode the initial value of the counter, denoted as  $n_0$ . In this example,  $n_0 = 4$ . The cap gadgets (of various sizes) are shown above the seed row as a combination of green and yellow tiles. The number of cap tiles is  $x$ .



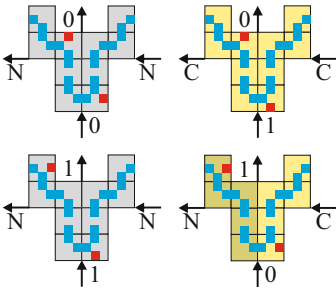
**Fig. 4.** Copy rows copy the bits advertised on the north side of the previous increment (or seed) row up for the next increment row. Copy rows encode each bit according to the “mirror-image” of the encoding utilized by the seed and increment rows. The pink square represents a negative glue that may or may not be present depending on whether or not the copy row is the first copy row to appear in the counter. These negative glues are used initiate the self-assembly of the second binary counter.

**The Copy Rows.** Copy rows self-assemble on top of increment rows (including the seed row) from left to right and have horizontal extent  $K$  (the actual width of the counter). Copy rows consist of a sequence of bit gadgets that utilize geometry



and the careful placement of the unique negative (red) glue in order to emulate cooperations (see Figure 2). In our construction, we have one bit gadget for every bit in the binary representation of  $n_0$  (this information is encoded directly into the bit gadget so that it knows which bit it is, e.g., most significant, least significant, third, etc). The bit gadgets that comprise each copy row are shown in Figure 4. In our construction, if a copy row reads a string of 1 bits, i.e.,  $2^m - 1$  for some  $m \in \mathbb{N}$ , it will terminate the counter and allow the cap gadget to attach.

**The Increment Rows.** Each increment row increments the value of the counter by 1. Increment rows self-assemble from right to left (compared to left to right for copy rows—hence the zig-zag nature of our counter). Similar to copy rows, increment rows consist of a sequence of (a different type of) bit gadgets that each know “which” bit they represent. The bit gadgets for increment rows are shown in Figure 5.



**Fig. 5.** The increment row bit gadgets read the bits of the previous copy row. The bit gadgets for increment rows emulate the standard binary counter tile types, such as those depicted in Figure 1 of [21]. For each bit in the binary representation of  $n_0$ , we have four types of bit gadgets. The inputs are always south (0 or 1 bit value) and east (carry/no-carry).

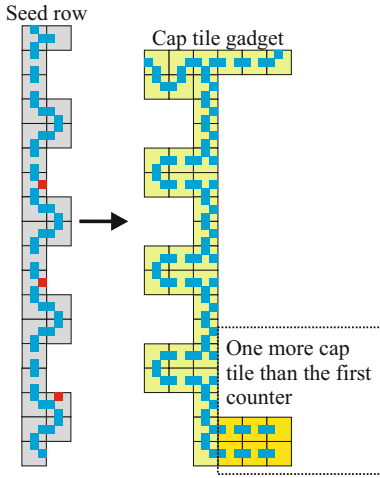
copy row. A south-facing negative glue tells the green path of tiles to “keep going.” Only the black seed tile type has an east-facing negative glue, which tells the path to “stop” and build the seed row for the second (horizontal) counter. Note that we do not encode any location information into these green tiles that crawl down the right side of the first counter, which means that there are  $O(1)$  such tiles.

The second binary counter (the base of the ‘U’ backbone) behaves similarly to the first counter except its top (logically, its least significant bit) is completely

The value of the final increment row in our counter is  $2^k - 1$  giving the counter an actual height of  $5(n - \lceil \frac{k}{5} \rceil)$  rows of tiles. The bit pattern of  $2^k - 1$  is detected by the (final) copy row, which terminates the counting. On top of the final copy row of the counter, a special cap tile gadget attaches, which is a path of tiles that fills in—and smooths out—the top of the jagged zig-zag counter. For each value of  $x$  (ranging from 1 to 9), we use a different cap tile gadget. The top portion of Figure 3 shows the two types of cap gadgets that we use in our construction—one allows additional “comb teeth” (each of varying height/length depending on  $x$ ) to attach and the other that simply caps the counter.

**Completing the Square.** After—and only after—the first binary counter completes, may the construction proceed. To the upper right corner of the first binary counter, a green path of tiles crawls down along the right side of the counter toward the seed tile. This green path of tiles detects the seed tile via the south-facing negative glue in the lower rightmost tile in each orange

smooth so as to allow the seed row of the third and final (vertical) binary counter to attach. Furthermore, the cap tile gadget for the second counter places one more row of cap tiles on top of (actually, to the right of) the second counter than the cap tile gadget did for the first counter to ensure that the terminal structure is a square (see Figure for an example). The cap tile gadget for the second counter also initiates the self-assembly of the seed row for the third binary counter (see Figure 6).



**Fig. 6.** The cap tile gadget ensures that there is one additional row of cap tiles to compensate for the relative positions of the first two counters

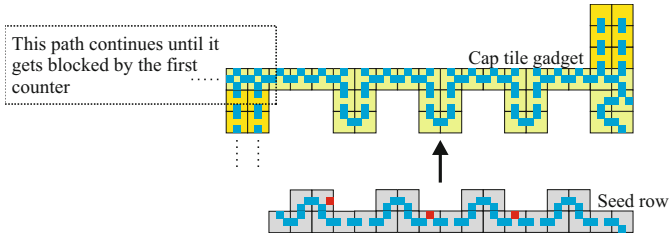
The third (and final) vertical counter in our construction completes the ‘U’-shaped backbone of the construction. This counter behaves similar to the first counter except its right edge is completely smooth. We also use a third type of cap tile gadget to form the smooth top of the square. This third type of cap gadget allows comb teeth (whose size depends on  $x$ ) to bind to its north side and also shoots a path of green tiles off to the left and back toward the first counter.

This green path of tiles is eventually blocked by the first counter, but as this path self-assembles to the left, it allows yellow filler tiles to fill in the interior of the square (see Figure 7 for an example) and comb teeth to attach on top.

This green path of tiles is eventually blocked by the first counter, but as this path self-assembles to the left, it allows yellow filler tiles to fill in the interior of the square (see Figure 7 for an example) and comb teeth to attach on top.

**Tile Complexity.** We use  $O(1)$  yellow filler tiles that either attach on top of (or to the right of) cap rows or fill in the interior of the square. There are  $O(1)$  green tiles that crawl down the right side of the first binary counter. The yellow tiles that fill in the bottom left corner of the square must grow to length  $O(K) = O(k)$  and stop for which  $O(k)$  tile types suffice. Finally, we must encode the appropriate bit location into every bit gadget of the seed row and every copy, increment and cap tile gadget. Since there are  $O(1)$  types of bit gadgets for each row and  $k$  bit locations in each of the three different counters that we use, the tile complexity of our construction is dominated by  $O(k) = O(\log N)$ .

It is interesting to note that our construction is essentially a spanning tree, much like the “ $2N - 1$ ” construction of [21]. However, in our construction, we are allowed to use a single negative glue type, which—in conjunction with some clever use of geometry—allows us to emulate the cooperativity of “temperature 2” self-assembly. Furthermore, the longest simple path of tiles in Rothemund and Winfree’s “ $2N - 1$ ” construction is  $2N - 1 = O(N)$  whereas the longest simple path of tiles in our construction is  $O(N \log N)$  but our construction ensures that the length of every simple (un-blocked) path of tiles cannot exceed  $O(\log N)$  without encountering the negative glue.



**Fig. 7.** The cap gadget for the third counter shoots a path of green tiles in the direction of—and is blocked by—the first counter. This green off-shoot not only allows any necessary cap tiles to attach on its top but initiates the self-assembly of the interior of the square via yellow filler tiles.

### 4 Turing Universality

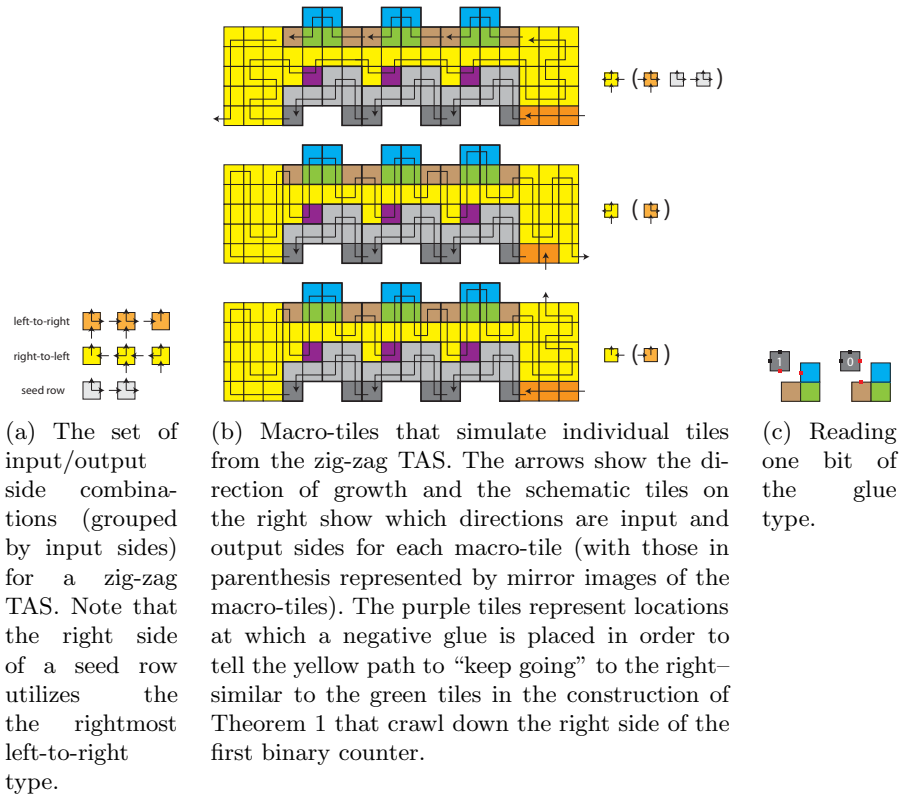
In this section, we show that for every zig-zag TAS, there is a restricted tile assembly system in the rgTAM that simulates it.

**Theorem 2.** *For every, zig-zag TAS  $\mathcal{T} = (T, \sigma, 2)$ , there exists a restricted TAS  $\mathcal{S} = (S, \gamma, 1)$  such that  $\mathcal{S}$  path simulates  $\mathcal{T}$  at scale factor  $O(\log |T|)$  with  $|S| = O(|T|)$ .*

Our construction for Theorem 2 is very similar in spirit to [2] in that we are essentially simulating a path using macro-tiles with geometrical bumps and dents except our construction uses negative glues to prevent erroneous growth. The remainder of this section is devoted to a brief, intuitive sketch of our construction for Theorem 2.

Intuitively,  $\mathcal{S}$  simulates  $\mathcal{T}$  by logically converting each tile type  $t \in T$  into a group (path) of tile types in  $S$  that self-assemble into a *macro-tile*. Let  $G$  be the number of unique strength-1 north/south glues in  $T$ . Each macro-tile is a path of  $2 \lceil \log G \rceil + 30$  tiles and forms in its entirety before allowing the next macro-tile to form, whence the scale factor in our construction is  $O(\log G) = O(\log |T|)$ . Moreover, we can use the fact that macro-tiles are all the same size in order to easily compute the indices  $i_{-1} < i_0 < \dots < i_k$  in the definition of path simulate.

As shown in Figure 8a, each  $t \in T$  has well-defined “input” and “output” sides (for convenience, and without loss of generality, we fix the seed row as growing from left to right), some of which may be the empty glue. The corresponding macro-tiles are depicted in Figure 8b. We encode each north/south glue in  $T$  as a unique  $(G + 1)$ -bit binary string (we also encode the empty glue label, whence each glue is represented as a  $(G + 1)$ -bit binary string). We then encode each binary string (representing a glue) as a path of bumps and dents along the north and south side of the appropriate macro-tile(s). Into the bumps and dents, we carefully place the negative glue type to either represent a ‘0’ or a ‘1’ bit—similar to the construction for Theorem 1. Note that we do not represent the “east/west” or strength-2 “north/south” glues of  $T$  in this manner



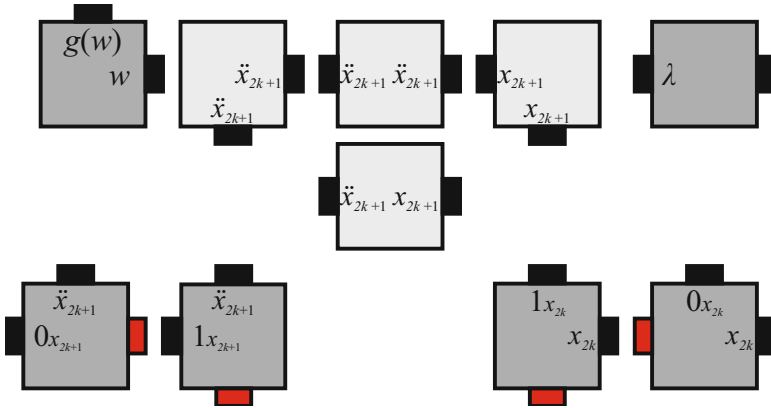
**Fig. 8.** Details of the zig-zag simulation construction

because in this case we encode these glue types in  $T$  on the glues of the tiles which serve as the beginning and ends (inputs and outputs) of the paths forming the corresponding macro-tiles.

A macro-tile that represents a tile type  $t \in T$  that binds via two “input” sides (e.g., south-east/south-west) self-assembles in two logical stages: reading the input glues and unpacking the output glues. In what follows, we will discuss the macro-tiles that represent tiles that have south/east input sides. The macro-tiles that emulate tiles with south/west input sides are constructed similarly.

**Reading the Input Glues.** In the first stage of the self-assembly of a “south-east input” macro-tile, an initial portion of its path crawls (either to the left or to the right) across the top of an existing macro tile. In doing so, the growing macro-tile path “reads” in, via a series of appropriate-placements of the negative glue in bumps and dents, a binary string, which represents a glue type in  $T$ . The method of “reading” a bit is depicted in Figure 8c and is similar to the technique used in Theorem 1 (see also Figure 2).

For each  $0 \leq i < G + 1$ , we have a group of tile types that are responsible for collecting the  $i^{\text{th}}$  bit of a binary string as they assemble a path to the left while

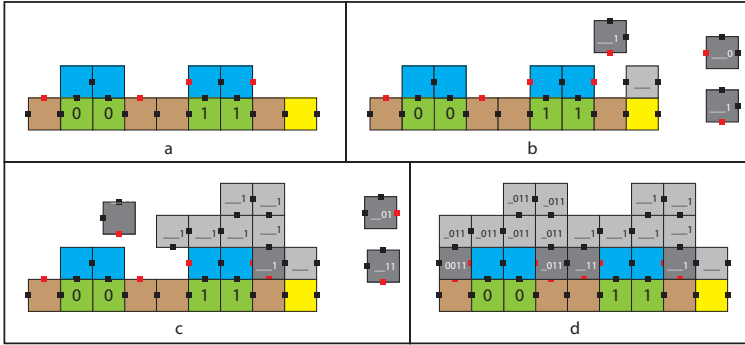


**Fig. 9.** The tile types that read a binary string from the top of an existing macro-tile as they self-assemble from right to left. For each  $i = 0, 1, \dots, G + 1$ , let  $x_i \in \{0, 1\}^i$ . Note that the east input glue is implicitly encoded into the tile types shown here. The upper right tile initiates the process of reading the input binary string. The upper left tile completes the process by mapping a pair of south-east input glues to the corresponding north-west output glues. Tiles for south-west input macro tiles are designed similarly.

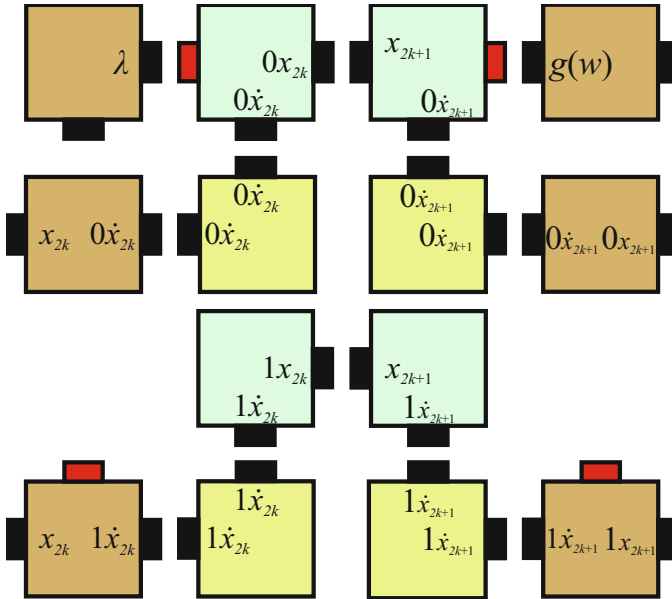
“remembering” the previous  $0 \leq j < i$  bits (see Figure 10). Note that not every tile type in the group that reads the  $i^{\text{th}}$  bit needs to remember *all*  $G + 1$  bits. In fact, it suffices for the group of tiles responsible for reading the  $i^{\text{th}}$  bit to only remember  $i$  bits. In order to do this, we use  $O(2^0 + 2^1 + \dots + 2^i) = O(2^{i+1})$  unique tile types, i.e.,  $O(1)$  tile types for each of the  $2^j$   $j$ -bit binary strings, whence we must have a total of  $O(2^{\log(G+1)}) = O(G) = O(|T|)$  unique tile types to read a glue from the top of an existing macro-tile (these tile types for a south-east input, north-west output macro-tile are shown in Figure 9). Once all  $G + 1$  bits have been collected, we have a group of  $O(G)$  unique tile types to convert the east input glue, along with the south input glue, into the appropriate output glue(s) for the macro-tile..

**Unpacking the Output Glues.** After the output glue(s) of a macro-tile have been determined, the macro-tile path crawls back across itself and determines when to “stop” via the purple tiles in Figure 8(b). Then the path crawls, once again, back across itself and “unpacks” the north output glue (it does not have to unpack the west output glue by the way we encode the east/west glue types in  $T$  in macro-tiles). We accomplish this task in a manner that is similar to—but essentially the opposite of—reading in a  $(G + 1)$ -bit binary string. To do this, we use  $O(G) = O(|T|)$  unique tile types (these tile types for a south-east input, north-west output macro tile are shown in Figure 11).

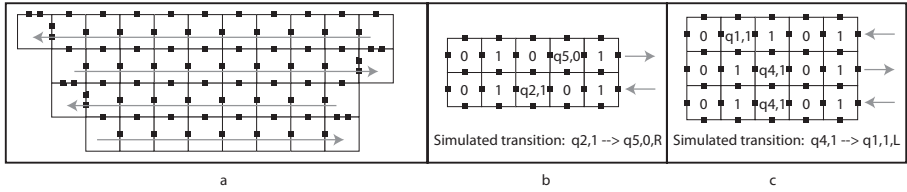
Finally, a macro-tile in  $S$  that represents a tile type  $t \in T$  that binds via a single, strength-2, input(output) side does not perform any input reading or output unpacking because we encode each strength-2 glue in  $T$  as a unique strength-1 glue in  $S$ . Thus, when such a macro tile self-assembles, it does so in a single logical stage. Our second main (universality) result is as follows.



**Fig. 10.** An example depicting the north side of a macro-tile being “read” by the south side of another macro-tile. Here, the binary number being read is “0011”. The northern macro-tile grows from right to left. Initially it has no information about the simulated glue to the south, and as it passes each position representing a bit, due to the configurations of the negative glue (pictured in red), it is able to place only one of two tiles, thus reading either a 0 or 1. See Figures 9 and 11 for more detail.



**Fig. 11.** The tile types that unpack a glue type into binary string as they self-assemble from right to left. For each  $i = 0, 1, \dots, G + 1$ , let  $x_i \in \{0, 1\}^i$ . Note that the east input glue is implicitly encoded into the tile types shown here. The upper right tile initiates the process of unpacking the input binary string. The upper left tile marks the completion of this process. Tiles for south-west input macro tiles are designed similarly.



**Fig. 12.** Sketch of a zig-zag Turing machine. (a) Rows grow in alternating directions (grey arrows) and are extended in width by one tile per row. Upward growth occurs only at the end of each row. (b) Example of a TM transition which moves the head to the right occurring in a row growing left-to-right. (c) Example of a TM transition which moves the head to the left. Note that if this transition is encountered by a row which is growing left-to-right, the transition will be skipped in that row (an effective “no op”), and instead simulated by the next row which grows right-to-left.

**Theorem 3.** *The rgTAM is Turing universal.*

*Proof.* For every Turing machine  $M$  and  $w \in \Sigma^*$ , there exists a zig-zag TAS  $\mathcal{T}_{M(w)}$  that simulates  $M$  on input  $w$  [5]. The basic idea is to design  $\mathcal{T}_{M(w)}$  so that self-assembly proceeds in a “zig-zag” growth pattern. This means that self-assembly proceeds according to a unique assembly sequence, which builds horizontal rows of tiles (configurations of  $M$ ) one at a time, alternating growth from left-to-right and right-to-left. Figure 12 shows an example of a zig-zag Turing machine construction.

By Theorem 2, we can simulate  $\mathcal{T}_{M(w)}$  with a restricted TAS in the rgTAM.

## References

1. Adleman, L., Cheng, Q., Goel, A., Huang, M.-D.: Running time and program size for self-assembled squares. In: Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing, pp. 740–748. ACM, New York (2001)
2. Adleman, L.M., Kari, J., Kari, L., Reishus, D., Sosik, P.: The undecidability of the infinite ribbon problem: Implications for computing by self-assembly. *SIAM Journal on Computing* 38(6), 2356–2381 (2009)
3. Barish, R.D., Schulman, R., Rothemund, P.W., Winfree, E.: An information-bearing seed for nucleating algorithmic self-assembly. *Proceedings of the National Academy of Sciences* 106(15), 6054–6059 (2009)
4. Chen, H.-L., Schulman, R., Goel, A., Winfree, E.: Reducing facet nucleation during algorithmic self-assembly. *Nano Letters* 7(9), 2913–2919 (2007)
5. Cook, M., Fu, Y., Schweller, R.: Temperature 1 self-assembly: Deterministic assembly in 3d and probabilistic assembly in 2d. In: Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms (2011)
6. Doty, D.: Randomized self-assembly for exact shapes. *SIAM Journal on Computing* 39(8), 3521–3552 (2010)
7. Doty, D., Kari, L., Masson, B.: Negative interactions in irreversible self-assembly. In: Sakakibara, Y., Mi, Y. (eds.) *DNA 16 2010*. LNCS, vol. 6518, pp. 37–48. Springer, Heidelberg (2011)

8. Doty, D., Lutz, J.H., Patitz, M.J., Summers, S.M., Woods, D.: Intrinsic universality in self-assembly. In: Proceedings of the 27th International Symposium on Theoretical Aspects of Computer Science, pp. 275–286 (2009)
9. Doty, D., Patitz, M.J., Reishus, D., Schweller, R.T., Summers, S.M.: Strong fault-tolerance for self-assembly with fuzzy temperature. In: Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS 2010), pp. 417–426 (2010)
10. Doty, D., Patitz, M.J., Summers, S.M.: Limitations of self-assembly at temperature 1. *Theoretical Computer Science* 412, 145–158 (2011)
11. Kao, M.-Y., Schweller, R.T.: Reducing tile complexity for self-assembly through temperature programming. In: Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2006), Miami, Florida, January 2006, pp. 571–580 (2007)
12. Kao, M.-Y., Schweller, R.T.: Randomized self-assembly for approximate shapes. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfssdóttir, A., Walukiewicz, I. (eds.) *ICALP 2008, Part I*. LNCS, vol. 5125, pp. 370–384. Springer, Heidelberg (2008)
13. Kinsella, J.M., Ivanisevic, A.: Enzymatic clipping of dna wires coated with magnetic nanoparticles. *Journal of the American Chemical Society* 127(10), 3276–3277 (2005)
14. Lathrop, J.I., Lutz, J.H., Summers, S.M.: Strict self-assembly of discrete Sierpinski triangles. *Theoretical Computer Science* 410, 384–405 (2009)
15. Luhrs, C.: Polyomino-safe dna self-assembly via block replacement. *DNA*, 112–126 (2008)
16. Majumder, U., Reif, J.: A framework for designing novel magnetic tiles capable of complex self-assemblies. In: Calude, C.S., Costa, J.F., Freund, R., Oswald, M., Rozenberg, G. (eds.) *UC 2008*. LNCS, vol. 5204, pp. 129–145. Springer, Heidelberg (2008)
17. Mao, C., Sun, W., Seeman, N.C.: Designed two-dimensional DNA holliday junction arrays visualized by atomic force microscopy. *Journal of the American Chemical Society* 121(23), 5437–5443 (1999)
18. Reif, J., Sahu, S., Yin, P.: Complexity of graph self-assembly in accretive systems and self-destructible systems. In: Carbone, A., Pierce, N.A. (eds.) *DNA 2005*. LNCS, vol. 3892, pp. 257–274. Springer, Heidelberg (2006)
19. Rickwood, D., Lund, V.: Attachment of dna and oligonucleotides to magnetic particles: methods and applications. *Fresenius' Journal of Analytical Chemistry* 330, 330–330 (1988), doi:10.1007/BF00469247
20. Rothmund, P.W.K.: Theory and experiments in algorithmic self-assembly, Ph.D. thesis, University of Southern California (December 2001)
21. Rothmund, P.W.K., Winfree, E.: The program-size complexity of self-assembled squares (extended abstract). In: *STOC 2000: Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, Portland, Oregon, United States, pp. 459–468. ACM, New York (2000)
22. Rothmund, P.W.K., Papadakis, N., Winfree, E.: Algorithmic self-assembly of DNA Sierpinski triangles. *PLoS Biology* 2(12), 2041–2053 (2004)
23. Soloveichik, D., Winfree, E.: Complexity of self-assembled shapes. *SIAM Journal on Computing* 36(6), 1544–1569 (2007)
24. Winfree, E.: Algorithmic self-assembly of DNA, Ph.D. thesis, California Institute of Technology (June 1998)
25. Winfree, E., Liu, F., Wenzler, L.A., Seeman, N.C.: Design and self-assembly of two-dimensional DNA crystals. *Nature* 394(6693), 539–544 (1998)



# Autonomous Resolution Based on DNA Strand Displacement

Alfonso Rodríguez-Patón<sup>1</sup>, Iñaki Sainz de Murieta<sup>1</sup>, and Petr Sosik<sup>1,2</sup>

<sup>1</sup> Departamento de Inteligencia Artificial,  
Universidad Politécnica de Madrid (UPM),  
Campus de Montegancedo s/n, Boadilla del Monte 28660 Madrid, Spain  
`inaki.sainzdemurieta@gmail.com`, `arpaton@fi.upm.es`

<sup>2</sup> Institute of Computer Science, Silesian University  
74601 Opava, Czech Republic  
`petr.sosik@fpf.slu.cz`

**Abstract.** We present a computing model based on the technique of DNA strand displacement which performs a chain of logical resolutions with logical formulae in conjunctive normal form. The model is enzyme-free and autonomous. Each clause of a formula is encoded in a separate DNA molecule: propositions are encoded assigning a strand to each proposition  $p$ , and its complementary strand to the proposition  $\neg p$ ; clauses are encoded comprising different propositions in the same strand. The model allows to run logic programs composed of Horn clauses by cascading resolution steps and, therefore, possibly function as an autonomous programmable nano-device. This technique can be also used to solve SAT. The resulting SAT algorithm has a linear time complexity in the number of resolution steps, whereas its spatial complexity is exponential in the number of variables of the formula.

## 1 Introduction

A construction of programmable biomolecular devices is one of the ultimate goals on the crossroad of biotechnology, nanotechnology and computer science. Their applications proposed so far in the literature include, among others, genetic engineering [1], biomedicine and drug delivery [2] and programmed molecule pattern formation [3]. Besides the input and output interface, i.e., the ability to sense and act in the complex inter- or intracellular environment, the key part of such a device is its internal logic controlling its behavior. Promising solutions for a physical implementation of logical operations with biomolecules were proposed in the area of DNA computing. After initial years when the discipline studied mainly combinatorial problems, another research line emerged, focusing on molecular logic for automated nano-devices. We cite only a few contributions to this broad topic, like for example the design of logic gates [4,5], DNA automata [6], as well as theoretical models [7]. These selected papers share a common property: the use of competitive hybridization commonly known as DNA strand displacement. Two complementary single strands of nucleic acids are joined to form a double helix under conditions of suitable temperature and pH.

A further step in the DNA-based logic are computing models that implement logical inference. Several results demonstrating theoretical principles of DNA-based inference were previously published, such as [8] based on the technique of Whiplash PCR, [9] applying DNA self-assembly, or [10] which used plasmid molecules and which presented also experimental results using human-assisted protocol. Recently, in the work of Shapiro et al. in [11] applying a similar automaton concept like the one previously proposed in [12], the authors developed and tested a system able to perform autonomously simple logical deductions with DNA molecules based on hairpin manipulation and unfolding. A few months later, Rodríguez-Patón et al. [13] presented a brief sketch of another inference model with DNA.

This article presents a DNA computing model based on strand displacement of four-way junctions [14,15], that encodes logical formulae in conjunctive normal form (CNF) and autonomously perform logical resolution of their clauses. Unlike [11], the model presented here is enzyme-free and its implementation can be built on the general protocol derived in [14,15]. Furthermore, as strand displacement models demonstrate special qualities [16] described in detail in the next section, our solution is partly scalable, time-responsive and energy-efficient.

We also demonstrate that our model is theoretically capable of solving the boolean satisfiability problem (SAT). The SAT problem continued to be approached in DNA computing after Lipton's work [17]. Most of the subsequent proposals have still followed the approach of generating all the possible solutions and then select the right ones using laboratory operations; improvements were introduced changing the way the initial set of solutions is encoded, reducing the number of laboratory steps [18,19,20]. Others have adapted classical SAT resolution algorithms like Davis-Putnam [21,22] into a DNA algorithm, following a constructive solution generation (instead of generating all and then selecting the right ones). Here we propose an alternative approach: we encode the full formula into DNA molecules representing clauses and let the system massively perform parallel logical resolution operations between the clauses.

The rest of the paper is structured as follows: Section 2 includes a description of the DNA strand displacement process, a quick review of the main concepts in propositional logic and the principles of the proposed DNA model. Section 3 describes how the model performs the autonomous resolution. Section 4 shows how the parallel application of multiple resolution steps, in combination with a few laboratory operations, can solve the SAT Problem. Finally, Section 5 summarizes the conclusions and future work.

## 2 Principles of the Model

Basic concepts of propositional logic used throughout the article are summarized first:

**Proposition.** The minimal syntactic entity in propositional logic. A proposition  $P$  can take values *True* or *False*.  $P = \text{True}$  is represented as  $P$ , whilst  $P = \text{False}$  is represented as  $\neg P$ .

**Formula.** A formula in propositional logic can be recursively defined as follows:

- Each propositional variable is a formula.
- If  $\varphi$  is a formula, then  $\neg\varphi$  is a formula.
- If  $\varphi$  and  $\Psi$  are formulas and  $\circ$  is any binary connective, then  $(\varphi \circ \Psi)$  is a formula. Here  $\circ$  could be  $\vee$ ,  $\wedge$ ,  $\rightarrow$ , or  $\leftrightarrow$ .

**Clause.** A logical formula consisting of a disjunction of propositions.

**Conjunctive Normal Form (CNF).** A logical formula is defined in conjunctive normal form *iff* it is expressed as a conjunction of clauses.

**Logical Resolution.** If  $C$  and  $C'$  are clauses and  $p$  is a proposition, then any assignment that satisfies both  $(C \vee p)$  and  $(C' \vee \neg p)$  also satisfies  $(C \vee C')$ .

**Hypothetical Syllogism.** Inference rule stating the following: if we have  $A \rightarrow B$  and  $B \rightarrow C$ , we can deduce  $A \rightarrow C$ .

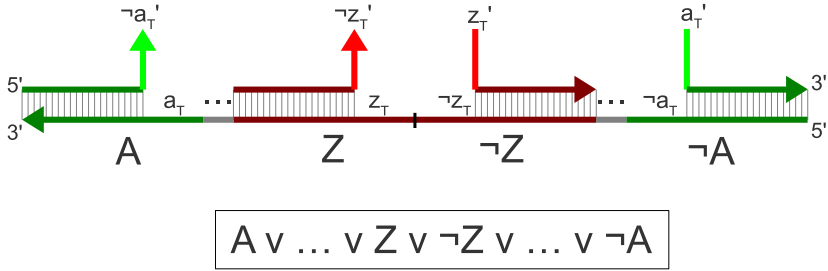
**SAT.** The Boolean Satisfiability Problem is the problem of determining whether the variables of a given formula can be assigned values in such a way as to make the formula evaluate to *True*.

## Encoding

Our encoding is based on the idea that every proposition is assigned a single nucleotide sequence denoting that  $A = \textit{True}$ , whereas its logical complement denoting  $A = \textit{False}$  (or  $\neg A$ ) is encoded with the complementary nucleotide sequence. For example, if the proposition  $A = \textit{True}$  is encoded with the DNA sequence  $5' - \textit{ATAAGG} - 3'$ , then  $A = \textit{False}$  will be encoded with its complementary strand  $3' - \textit{TATTCC} - 5'$ . This dual encoding feature allows the logical resolution to be applied to different clauses in an autonomous way. The same encoding principle is used, e.g., also in [13,18].

Clauses in conjunctive normal form (CNF) can be encoded comprising these propositions into the same DNA strand, following the rules below (see Fig. 1):

- Propositions are named with capital letters from  $A$  to  $Z$ . Propositions belonging to the same clause are encoded in the same DNA strand and sorted as follows: positive propositions in ascending order, starting from the 3' end of the strand; then negative propositions in descending order, finishing at the 5' end.
- Each proposition is annealed to a “cover” strand of same length with the following structure:
  - A toehold region will be located at the 3' end when the cover anneals to an affirmed proposition (E.g.  $\neg a'_T$  in Fig. 1), or at the 5' end when the cover anneals to a negated proposition (E.g.  $z'_T$  in Fig. 1). In both cases the toehold stays single stranded and does not anneal to the proposition strand.
  - In affirmed propositions, the cover strand anneals its 5' non-toehold region with the 3' region of the proposition strand. A free toehold region stays single stranded at the 5' end of the proposition strand (E.g.  $a_T$  in Fig. 1).



**Fig. 1.** The DNA encoding of clauses with toehold regions

- In negated propositions, the cover strand anneals its 3' non-toehold region with the 3' region of the proposition strand. A free toehold region stays single stranded at the 3' end of the proposition strand (E.g.  $\neg z_T$  in Fig. 1).
  - The cover strand of a given proposition  $P$  must be complementary to the cover strand of  $\neg P$ .
- Each cover strand is labeled with a fluorophore reporter. It does not play any role in the resolution mechanism but it will be required by the next steps of the SAT resolution algorithm.

Concerning the detailed physical implementation of these abstract clause-encoding sequences, there already exist several experimental implementations of molecular logic based on strand displacement so that we can to a large extent rely on the obtained results and properties. The pioneering work of Seelig et al. [4] presented the construction of enzyme-free DNA logic gates based on strand displacement. Their “dual-rail” representation can be considered as a base for the physical encoding in our design in the sense that both a logical proposition and its negation are represented by the presence of a certain DNA strand specie; in our case, however, these species are mutually complementary. Later, building on the recent and general work [23], the article [16] introduced an improved design of molecular circuits with the following declared properties: scalable, time-responsive, energy-efficient and digital. We briefly summarize these properties and note how they apply to our construction.

**scalable:** *The preparation of the DNA strands can be done in a single test tube instead of a separate tube for each DNA specie.* This is only partly true in our case: all cover strands with toeholds encoding propositions can be divided into two groups, positive and negative, and each group can be prepared together in a single test tube. However, the long DNA strands encoding clauses of the initial program must be prepared separately since they may contain mutually complementary parts.

**time-responsive:** *After some initial computation, the system is still able to react to changes of the input and accordingly re-compute the output.* In our design, some initial resolution steps can proceed in the mixture of DNA

species encoding clauses, but when new species encoding new facts or rules are added, new resolution steps will start, hence the system responds. However, whenever there are enough facts for refutation, then the specie corresponding to the empty clause is irreversibly produced.

**energy-efficient:** *If inputs are given ideally (i.e., with 0 concentration of unwanted species), then the system converts into steady state when outputs are also ideal and the system is in an energetic equilibrium; this is true in our design, too.*

**digital:** *The DNA reactions are able to perform signal restoration when the concentration of DNA species encoding specific logic information is low, to obtain a digital abstraction of analog concentration values.* Such a property is the subject of further research in our case since a large number of intermediate clauses can emerge during the resolution and some kind of “universal signal restoration gate” would be needed.

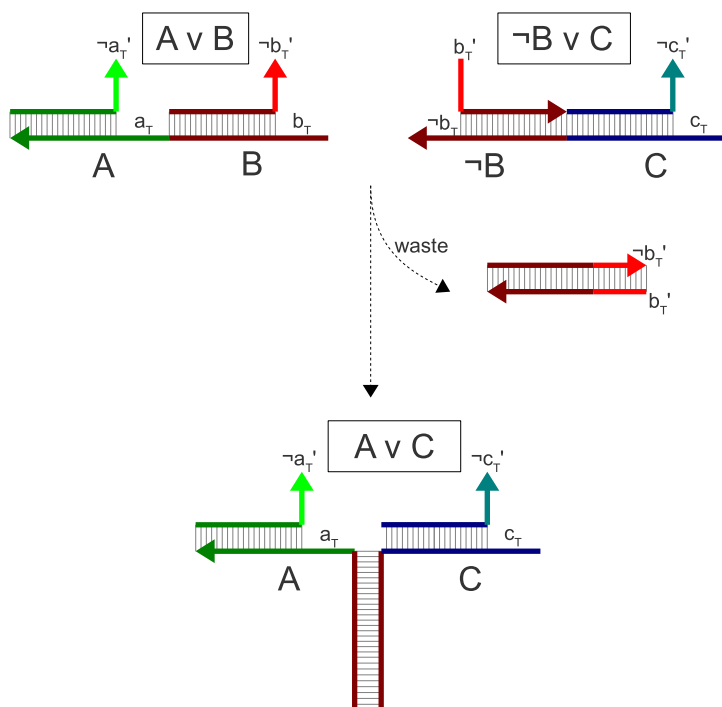
### 3 Autonomous Resolution Using DNA

The resolution is perhaps the most studied propositional refutation system due to its simplicity and importance in many automatic theorem proving procedures [24]. The resolution principle states that, if  $C$  and  $C'$  are clauses and  $p$  is a proposition, then any assignment that satisfies both  $(C \vee p)$  and  $(C' \vee \neg p)$  also satisfies  $(C \vee C')$ . Consequently,  $(C \vee p)$  and  $(C' \vee \neg p)$  is unsatisfiable if and only if  $(C \vee C')$  is unsatisfiable. Having a look at Fig. 2, we can see how resolution is automatically performed between two clauses encoded as described in Section 2: the molecules encoding clauses  $A \vee B$  and  $\neg B \vee C$  merge together into another molecule that encodes the resolvent clause,  $A \vee C$ , which can be later used in further resolution iterations. Although the resolvent contains an extra double stranded region (formed by  $B$  and  $\neg B$ ), it has no free toehold and cannot alter computations reacting with other clauses.

The example depicted in Fig. 3 shows a resolution reaction of two unary clauses,  $B$  and  $\neg B$ , in detail.

- In the first step, the toeholds of both cover strands anneal ( $\neg b'_T$  with  $b'_T$ ), and so do the toeholds of the proposition strands ( $\neg b'_T$  with  $b'_T$ ). A *Holliday junction* [25,26] is formed as result of both hybridizations.
- At this point, the junction starts a spontaneous branch migration in a random walk [14,15].
- When the Holliday junction reaches the distal end of the double stranded region, the strand exchange completes producing two stable duplexes: one of them made with the cover strands, the other one with  $B$  and  $\neg B$ .

At the end of the process,  $B$  and  $\neg B$  form a fully hybridized double stranded molecule; unlike in Figure 2, it contains no single stranded segments that encode propositions, meaning that the formula  $B \wedge \neg B$  is not satisfiable. This resolution reaction has evolved into an empty clause, represented as  $\emptyset$ .



**Fig. 2.** Basic resolution step. The clauses  $A \vee B$  and  $\neg B \vee C$  are resolved on  $B$ , producing the resultant clause  $A \vee C$ .

This principle can be cascaded, as demonstrated in [23,16], resulting in more complex logic steps as the *hypothetical syllogism*: if we have  $A \rightarrow B$  and  $B \rightarrow C$ , we can deduce  $A \rightarrow C$ . As every implication can be formulated as a disjunction, we can rewrite the previous statement as follows: if we have  $\neg A \vee B$  and  $\neg B \vee C$ , we can deduce  $\neg A \vee C$ , which also matches the example of Figure 2. The model also allows multiple iterations of the hypothetical syllogism.

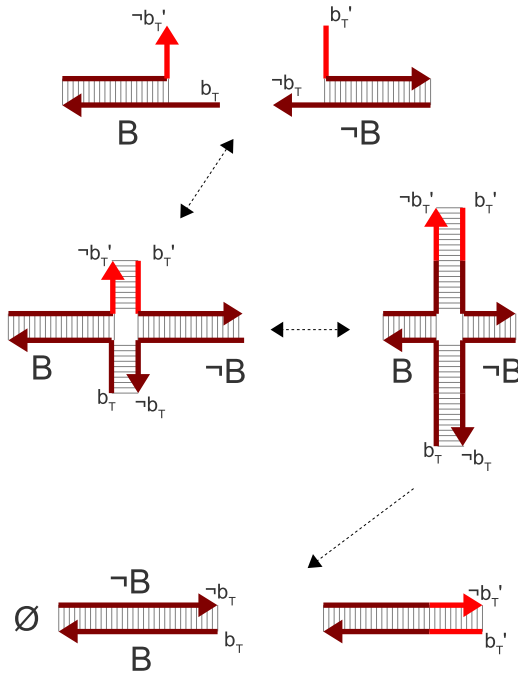
Finally, consider a logic program consisting of a sequence of Horn clauses in the form

$$(A_1 \wedge A_2 \wedge \dots \wedge A_n) \rightarrow B$$

which can be rewritten as

$$\neg A_1 \vee \neg A_2 \vee \dots \vee \neg A_n \vee B.$$

Each such clause (and, of course, each unary clause representing a fact) can be encoded as a separate DNA specie as described above and then the autonomous resolution can proceed, until the empty clause is eventually obtained, indicating that the refutation process was successful. A detection of the empty clause is described in the next section.



**Fig. 3.** 4-way branch migration process that implements the resolution step of two unary clauses  $B$  and  $\neg B$

## 4 Boolean Satisfiability Problem (SAT)

The problem of satisfiability of boolean formulae in CNF was the first decision problem proved to be NP-complete [27]. Given any boolean formula, we can rewrite it to CNF in a way that both of them are equisatisfiable, and then encode it using our model. Our algorithm will apply the resolution to all the clauses and propositions, looking for potential refutations to the initial formula.

A resolution refutation of a non-satisfiable CNF formula  $\varphi$  is a sequence of clauses  $C_1 \dots C_s$ , where each  $C_i$  is either a clause of  $\varphi$  or it is inferred from earlier clauses by the resolution rule, and  $C_s$  is the empty clause. If a refutation can be derived from an initial formula, then the formula is unsatisfiable. We can encode the initial formula following our DNA model, adding several copies of each molecules to allow all the possible resolution reactions to be performed in parallel.

### Identifying the Empty Clause

The question now is, how our model will determine whether the empty clause has been derived or not. For a better understanding of the process, let us review some simple cases to understand the general rule:

- Fig. 2 shows how our resolution model works for a satisfiable formula with two binary clauses:  $(A \vee B) \wedge (\neg B \vee C)$ . When trying to derive an empty clause, the logical resolution sequence would be the following<sup>1</sup>:

$(A \vee B), (\neg B \vee C)$

$(A \vee C)$

We can see the expected logical output matches the DNA output of Fig. 2. No refutation has been derived, thus the formula is satisfiable (SAT). At the end of the reaction, all the molecules contain at least one cover strand.

- Fig. 3 shows how resolution works for the simplest unsatisfiable formula:  $B \wedge \neg B$ . When trying to derive the empty clause, the logical resolution sequence would be the following:

$(B), (\neg B)$

$\emptyset$

The reaction product in Figure 3 is a molecule containing no cover strands, labeled as  $\emptyset$ . It represents the last clause of the empty clause refutation. Hence, the formula is unsatisfiable (UNSAT). The expected logical output matches the DNA output in Fig. 3.

- Fig. 4 shows another simple case of unsatisfiable formula:  $\neg A \wedge (A \vee B) \wedge \neg B$ . The logical resolution sequence would be the following:

$\neg A, (A \vee B), \neg B$

$A, \neg A$

$\emptyset$

As in Fig. 3, we can see a molecule containing no cover strands, labeled as  $\emptyset$ . It matches the expected logical output detailed in the previous lines and the formula is UNSAT.

- Fig. 5 shows a case of satisfiable formula:  $(A \vee B) \wedge (\neg B \vee \neg A)$ . When trying to derive an empty clause, the logical resolution sequence would be the following:

$(A \vee B), (\neg B \vee \neg A)$

$(A \vee \neg A)$

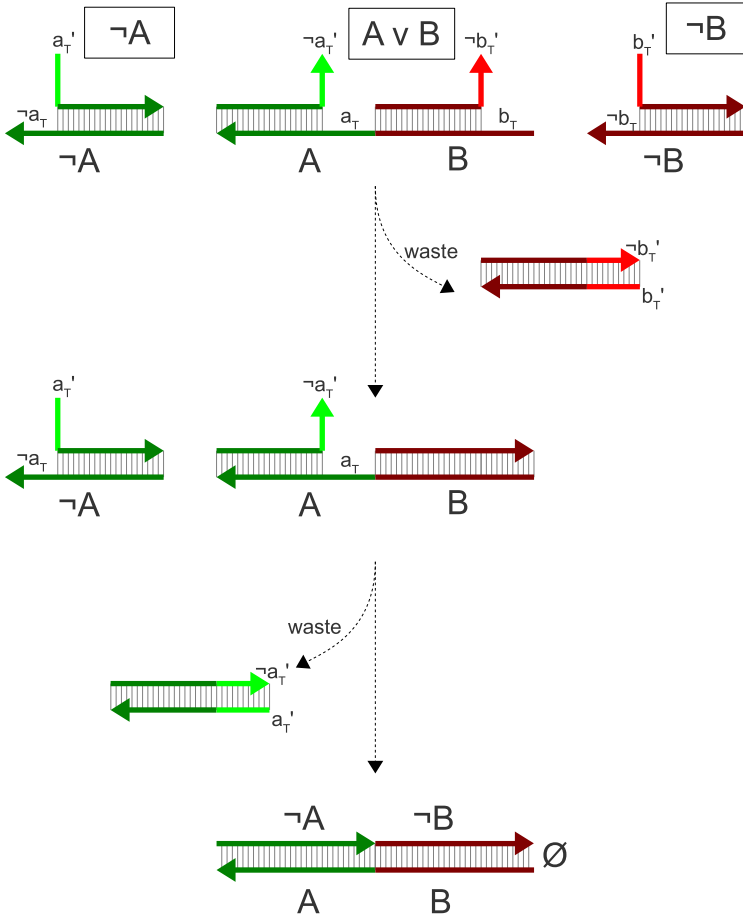
*TRUE*

Although we can find a molecule containing no cover strands, it does not represent an empty clause in this case. It is so because the second reaction depicted in Figure 5 does not correspond to a resolution operation since the annealed propositions belong to the same clause. This fact is still distinguishable looking at the structure of the final molecule (made of only cover strands): two DNA strands are annealed in more than one proposition. This differs from Fig. 4 where the empty clause molecule has a nick in the middle.

---

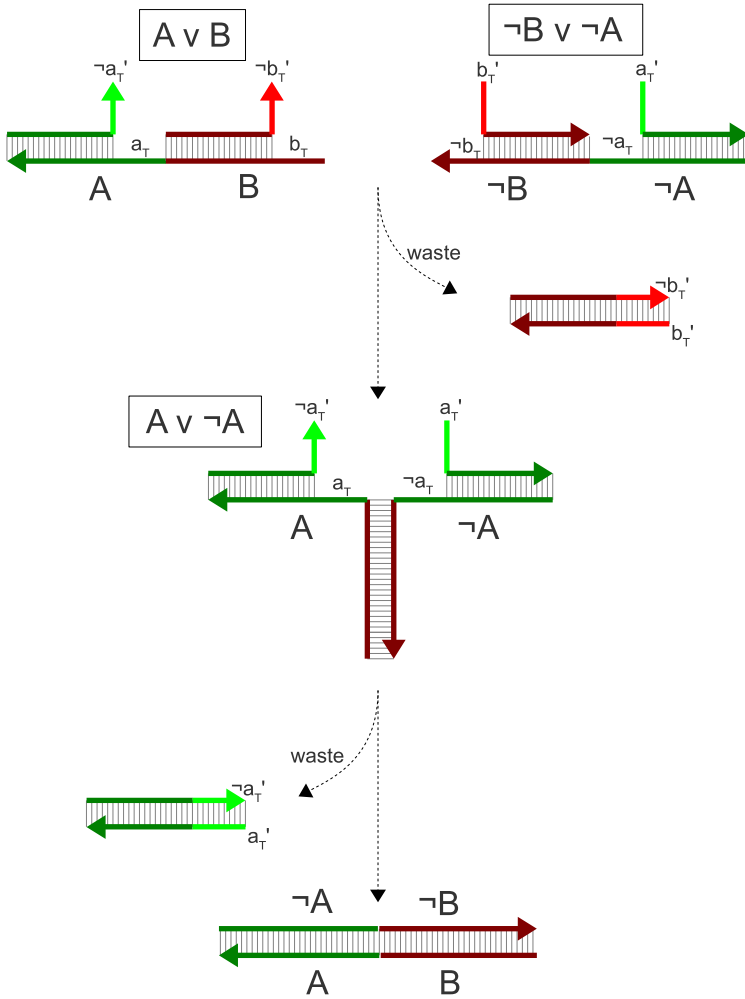
<sup>1</sup> The first clause of each line is the output of applying resolution at the previous line. Second clause of each line represents the new clause added to the sequence at that step. First line contains the initial clauses.





**Fig. 4.** *Unsatisfiable formula:*  $\neg A \wedge (A \vee B) \wedge \neg B$ . The first resolution between the clauses  $(A \vee B)$  and  $\neg B$  yields the clause  $A$ . In the second reaction,  $A$  and  $\neg A$  are resolved into the empty clause  $\emptyset$ , meaning the formula is UNSAT.

At this stage we should be able to determine the satisfiability of any formula by looking at the structure of output molecules composed only by non-cover strands (if any). These molecules are easily distinguishable since each cover strand contains a fluorescent marker. The empty clause corresponds to a non-fluorescent double-stranded molecule with nicks between all parts encoding propositions. However, there might be other molecules with the same length and structure, resulting from non-resolution reactions, see Fig. 5. Comparing with Fig. 4, one can see that the output molecules are almost identical except for the nick between  $\neg A$  and  $\neg B$  in Fig. 4, and if they occurred simultaneously, it would be impossible to distinguish them by electrophoresis.



**Fig. 5.** *Satisfiable formula:*  $(A \vee B) \wedge (\neg B \vee \neg A)$ . The first resolution between the clauses  $(A \vee B)$  and  $(\neg B \vee \neg A)$  yields the clause  $(A \vee \neg A)$  which is always *True*. Thus the second hybridization does not represent a resolution operation, since the propositions involved belong to the same clause, but the logic operation  $A \vee \neg A$  which is always true.

Therefore, we add to the 5' end of each non-cover strand a few extra nucleotides which will not bind to cover strands but instead it will form a single-stranded sticky end. Its purpose is to increase the weight/size of the molecule during electrophoresis and to allow the differentiation of these two cases. Denote  $n$  the length of non-cover strands encoding propositions and  $k$  the length of the extra single-stranded end. The empty clause produced by refutation of  $x$

propositions contains  $x$  double-stranded sequences, each of length  $n$ , since  $2xn$  nucleotides in total. Furthermore, it contains  $x + 1$  single-stranded sticky ends, each of the length  $k$ , hence  $(x + 1)k$  nucleotides in total. The sum is  $2xn + (x + 1)k = x(2n + k) + k$ .

By a proper choice of  $n$  and  $k$ , it can be prevented that another molecule encoding a non-empty clause would occupy the same place in the electrophoretic band, but a more thorough analysis will be provided in an extended version of the paper.

## DNA Algorithm

We outline, step by step, our model to apply resolution to a set of CNF clauses and determine whether the formula is satisfiable or not. Let  $n$  be the length of non-cover strands encoding propositions and  $k$  the length of sticky ends.

1. Encode the clauses as DNA molecules, according to Section 2 and Fig. 1.
2. Mix all the clauses in the same dissolution and let all the possible resolution reactions take place. If the formula is unsatisfiable, then the empty clause is produced after at most  $v$  reactions cycles,  $v$  being the number of variables (see the analysis bellow).
3. Filter the result using gel electrophoresis. If all the bands formed in the gel show fluorescence, it means no empty clause has been generated by any resolution reaction (as in Fig. 2), hence the formula is SAT. The process ends here.
4. Otherwise, if there are bands without fluorescence and at least one of them corresponds to the length of  $x(2n + k) + k$  nucleotides for  $x \geq 1$ , then the formula is UNSAT.

A quick analysis of time and space complexity of the algorithm can be provided:

- Step 1 (clause encoding) can be considered constant in time.
- Step 2 (massive parallel resolutions) would depend on the minimal number of parallel resolution steps to derive the empty clause.
- Step 3 (gel electrophoresis) is another constant time operation.

Concerning the minimal number of the parallel resolution steps, it is equal to the minimal depth of a refutation tree or, more generally, DAG (Directed Acyclic Graph) for a given formula since any such DAG is a part of the resolution performed by our algorithm. Assuming the regular tree resolution which is complete (see, e.g., [28] for definitions), the depth of the tree is at most the number of variables in the formula, see Theorem 3.1 in [28] and its proof. Therefore, the time complexity of the algorithm is  $\mathcal{O}(v)$  where  $v$  is the number of variables in the original formula.

The spatial complexity of the algorithm depends generally on its Step 2, more precisely on the size of the refutation, i.e., the number of occurrences of clauses. It was proven already in [29] that even the size of general refutation (the smallest refutation with no restrictions on the sequence of resolution steps) is exponential in the number of variables of the formula, establishing the lower bound for our

space complexity. Since our algorithm cannot generate more than all possible clauses on  $v$  variables whose number is  $3^v$  (each variable is either affirmed or negated or absent), the upper bound is asymptotically the same. Hence, the space complexity of the algorithm measured as the number of different DNA molecules is  $2^{\Theta(v)}$ .

## 5 Conclusions and Future Works

We have introduced a new DNA model for representation of logical formulas in CNF which contains an implicit resolution mechanism that acts on any pair of clauses containing complementary propositions. The model is based on the strand displacement techniques and the resolution is completely autonomous, except for the last detection step when a possibly human-assisted electrophoresis is performed. Our model is completely enzyme-free and its implementation can be based on experimentally verified and general design derived in [14,15]. According to the properties examined in [16], the model can be characterized as partly scalable, time-responsive and energy-efficient.

The first application of our model is the autonomous resolution including the hypothetical syllogisms as its special case, and, by cascading the resolution steps, the refutation proving technique for sentences in propositional logic. In this way, the model could perhaps become a part of a programmable nano-automaton sensitive to various inputs and running a logic program. The main drawback we see in this sense is the detection step based on the electrophoresis, when the output signal is analogue and its level can be low (it depends on the presence of molecules of certain lengths which might be possibly present in low concentrations).

Another application is the resolution of the SAT problem. We can theoretically resolve any instance of SAT by applying multiple refutation technique in parallel, trying to derive the empty clause. If no molecule is a potential candidate to be the empty clause, we can automatically answer that the problem is SAT. An interesting feature is that, if there are several refutations that derive the empty clause, all of them are “registered” in the output. By examining this we can also find the quickest way to determine the unsatisfiability.

This application to SAT is based on the assumption that all the original and resolvent clauses are available during the whole reaction process. In practice, however, some of them could be consumed during previous reactions. A deeper analysis is needed to study this problem and possibly also the completeness of an accordingly restricted form of refutation.

We have derived that our algorithm works in the time proportional to the number  $v$  of logical variables in the original formula. This upper bound also provides a stop condition for the Step 2 of the algorithm when after  $v$  steps the correct solution can be detected. In contrast, the number of copies of clauses to be processed (i.e., the number of distinct DNA molecules produced during the algorithm and defining its space complexity) grows exponentially with the number of variables in the formula.

There is room for improvement of the presented model. It would be highly desirable to achieve spatial conformations that significantly differ for the cases of the empty clause and double implication, which would eliminate the need of electrophoresis. We also need to include some kind of signal restoration at the output, to be able to react to the presence of output molecules in low concentrations. These two goals are mutually interconnected and they aim at making the output digital and autonomously connected to further biochemical processes. Another improvement opportunity is a deep analysis of how the empty clause refutation candidates are formed, checking how slight modifications in the way we build clauses and sort the literals can affect the structure of the candidate molecules.

**Acknowledgments.** Research was partially supported by project BACTOCOM funded by grant from the European Commission under the VII Framework Programme (FET Proactive area), by Comunidad de Madrid and UPM, by Spanish Ministerio de Ciencia e Innovación under project TIN2009-14421 and by the Silesian University in Opava under the project SGS/7/2011.

The article has been made in connection with project IT4Innovations Centre of Excellence, reg. no. CZ.1.05/1.1.00/02.0070 supported by Research and Development for Innovations Operational Programme financed by Structural Funds of Europe Union and from the means of state budget of the Czech Republic.

## References

1. Deans, T.L., Cantor, C.R., Collins, J.J.: A tunable genetic switch based on RNAi and repressor proteins for regulating gene expression in mammalian cells. *Cell* 130(2), 363–372 (2007)
2. Benenson, Y., Gil, B., Ben-Dor, U., Adar, R., Shapiro, E.: An autonomous molecular computer for logical control of gene expression. *Nature* 429, 423–429 (2004)
3. Basu, S., Gerchman, Y., Collins, C.H., Arnold, F.H., Weiss, R.: A synthetic multicellular system for programmed pattern formation. *Nature* 434(7037), 1130–1134 (2005)
4. Seelig, G., Soloveichik, D., Zhang, D.Y., Winfree, E.: Enzyme-free nucleic acid logic circuits. *Science* 314(5805), 1585–1588 (2006)
5. Frezza, B.M., Cockroft, S.L., Ghadiri, M.R.: Modular multi-level circuits from immobilized DNA-based logic gates. *J. Am. Chem. Soc.* 129(48), 14875–14879 (2007)
6. Takahashi, K., Yaegashi, S., Kameda, A., Hagiya, M.: Chain reaction systems based on loop dissociation of DNA. In: Carbone, A., Pierce, N.A. (eds.) *DNA 2005*. LNCS, vol. 3892, pp. 347–358. Springer, Heidelberg (2006)
7. Cardelli, L.: Strand Algebras for DNA Computing. In: Deaton, R., Suyama, A. (eds.) *DNA 15*. LNCS, vol. 5877, pp. 12–24. Springer, Heidelberg (2009)
8. Kobayashi, S.: Horn clause computation with DNA molecules. *J. Comb. Optim.* 3, 277–299 (1999)
9. Uejima, H., Hagiya, M., Kobayashi, S.: Horn clause computation by self-assembly of DNA molecules. In: Jonoska, N., Seeman, N.C. (eds.) *DNA 2001*. LNCS, vol. 2340, pp. 308–320. Springer, Heidelberg (2002)
10. Wasiewicz, P., Janczak, T., Mulawka, J.J., Plucienniczak, A.: The inference based on molecular computing. *Cybernetics and Systems: An International Journal* 31(3), 283–315 (2000)

11. Ran, T., Kaplan, S., Shapiro, E.: Molecular implementation of simple logic programs. *Nature Nanotechnology* 4(10), 642–648 (2009)
12. Benenson, Y., Paz-Elizur, T., Adar, R., Keinan, E., Livneh, Z., Shapiro, E.: Programmable and autonomous computing machine made of biomolecules. *Nature* 414(6862), 430–434 (2001)
13. Rodríguez-Patón, A., Larrea, J.M., Sainz de Murieta, I.: Inference with DNA molecules. In: Calude, C.S., Hagiya, M., Morita, K., Rozenberg, G., Timmis, J. (eds.) UC 2010. LNCS, vol. 6079, page 192. Springer, Heidelberg (2010)
14. Panyutin, I.G., Hsieh, P.: The kinetics of spontaneous dna branch migration. *Proceedings of the National Academy of Sciences* 91(6), 2021–2025 (1994)
15. Biswas, I., Yamamoto, A., Hsieh, P.: Branch migration through DNA sequence heterology. *Journal of Molecular Biology* 279(4), 795–806 (1998)
16. Chiniforooshan, E., Doty, D., Kari, L., Seki, S.: Scalable, time-responsive, digital, energy-efficient molecular circuits using DNA strand displacement. In: Sakakibara, Y., Mi, Y. (eds.) DNA 16 2010. LNCS, vol. 6518, pp. 25–36. Springer, Heidelberg (2011)
17. Lipton, R.J.: DNA solution of hard computational problems. *Science* 268(5210), 542–545 (1995)
18. Sakamoto, K., Gouzu, H., Komiya, K., Kiga, D., Yokoyama, S., Yokomori, T., Hagiya, M.: Molecular computation by DNA hairpin formation. *Science* 288(5469), 1223–1226 (2000)
19. Manca, V., Zandron, C.: A clause string DNA algorithm for SAT. In: Jonoska, N., Seeman, N.C. (eds.) DNA 2001. LNCS, vol. 2340, pp. 172–181. Springer, Heidelberg (2002)
20. Manca, V.: DNA and membrane algorithms for SAT. *Fundamenta Informaticae* 49(1), 205–221 (2002)
21. Ogihara, M.: Breadth first search 3SAT algorithms for DNA computers (1996)
22. Wang, X., Bao, Z., Hu, J., Wang, S., Zhan, A.: Solving the SAT problem using a DNA computing algorithm based on ligase chain reaction. *Biosystems* 91(1), 117–125 (2008)
23. Soloveichik, D., Seelig, G., Winfree, E.: DNA as a universal substrate for chemical kinetics. *Proceedings of the National Academy of Sciences* 107(12), 5393–5398 (2010)
24. Davis, M., Putnam, H.: A computing procedure for quantification theory. *J. ACM* 7(3), 201–215 (1960)
25. Stahl, F.W.: The Holliday junction on its thirtieth anniversary.. *Genetics* 138(2), 241–246 (1994)
26. Liu, Y., West, S.C.: Happy Hollidays: 40th anniversary of the Holliday junction. *Nature Reviews Molecular Cell Biology* 5(11), 937–944 (2004)
27. Cook, S.A.: The complexity of theorem-proving procedures. In: STOC 1971: Proceedings of the Third Annual ACM Symposium on Theory of Computing, New York, NY, USA, pp. 151–158 (1971)
28. Esteban, J.L., Torán, J.: Space Bounds for Resolution. *Information and Computation* 171(1), 84–97 (2001)
29. Haken, A.: The intractability of resolution. *Theoretical Computer Science* 39, 297–308 (1985)

# Multiple Molecular Spiders with a Single Localized Source—The One-Dimensional Case (Extended Abstract)

Oleg Semenov, Mark J. Olah, and Darko Stefanovic

Department of Computer Science, University of New Mexico

**Abstract.** Molecular spiders are nanoscale walkers made with DNA enzyme legs attached to a common body. They move over a surface of DNA substrates, cleaving them and leaving behind product DNA strands, which they are able to revisit. Simple one-dimensional models of spider motion show significant superdiffusive motion when the leg-substrate bindings are longer-lived than the leg-product bindings. This gives the spiders potential as a faster-than-diffusion transport mechanism. However, analysis shows that single-spider motion eventually decays into an ordinary diffusive motion, owing to the ever increasing size of the region of cleaved products. Inspired by cooperative behavior of natural molecular walkers, we propose a model for multiple walkers moving collectively over a one-dimensional lattice. We show that when walkers are sequentially released from the origin, the collective effect is to prevent the leading walkers from moving too far backwards. Hence there is an effective outward pressure on the leading walkers that keeps them moving superdiffusively for longer times, despite the growth of the product region.

## 1 Introduction

Molecular walkers are nanometer-sized molecules that move over surfaces with tracks of chemical sites by means of chemical reactions. They provide a means to transport chemicals by non-diffusive directed motion. Molecular walkers are ubiquitous as a transport mechanism in biological systems [12], and many of the complex regulatory cellular processes are controlled by the actions of molecular walkers such as kinesin and dynein [7]. It has been demonstrated experimentally that these cellular molecular walkers work in teams, wherein their collective action leads to behaviors not possible for a single walker [3]. In addition, theoretical models predict that collective cooperative or competitive behavior of walkers is fundamentally different from the behavior of individual walkers [8, 6, 5].

Inspired by the potential for walker cooperation, we propose a model describing the collective behavior of teams of molecular walkers. Our model is based on synthetic walkers called *molecular spiders* [10] (Sec. 2). Molecular spiders have two or more enzymatic legs attached to a common body. The legs are deoxyribozymes—catalytic sequences of single-stranded DNA that can cleave

complementary single-stranded DNA *substrates*. Spiders move over a surface coated with substrates, attaching to, cleaving, and detaching from the substrate sites. Spiders leave behind *product* strands, which are the lower portions of the cleaved surface-bound substrates. Experiments have shown that this mechanism allows spiders to move directionally over nanoscale tracks of regularly spaced DNA substrates [9].

Antal and Krapivsky proposed a simple abstract model that describes spider motion in one dimension (1D) as a continuous-time Markov process [2, 1]. We call it the AK model, and describe it in Sec. 2.1. In previous work, we showed via computer simulation and analytical arguments that walkers in the AK model can move superdiffusively over significant times and distances [13]. However, analysis shows that the AK walkers always eventually end up slowing down and moving as an ordinary diffusive process. This can be explained by observing that spiders move superdiffusively only when there is a difference in residency times between substrates and products. When a walker is not attached to any substrates, its motion is unbiased and diffusive. As a walker moves, it creates an increasingly large region of products that is difficult to escape from, and diffusing within it eventually consumes most of the walker’s time.

In this work, we propose that the collective action of many spiders simultaneously moving over a 1D surface can act to ameliorate the decay of the superdiffusive motion of certain (namely, leading) walkers. The exclusionary properties of spiders act to limit the effective size of the product sea, and prevent the furthestmost walkers from moving too far backwards. In Sec. 3 we describe a model for multiple spiders interacting on an infinite 1D lattice. Using Kinetic Monte Carlo methods, we show that multi-spider systems exhibit significantly superdiffusive motion within the time bounds studied (Sec. 4). These preliminary results indicate that multi-spider systems exhibit behavior not seen in single-spider systems, and this behavior has the potential to be used to perform useful tasks in nanoscale computational and communication systems by providing a faster-than-diffusion mechanism of transport.

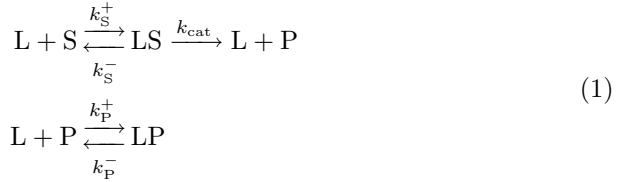
## 2 Molecular Spiders

Walkers in our model are nearly identical (except for a detail that arises only in multi-spider interactions, cf. Sec. 3) to the walkers of the AK model, which we summarize here (see Refs. [1, 13] for a complete treatment).

A molecular spider has a rigid, chemically inert body (such as streptavidin) and several flexible legs made of deoxyribozymes—enzymatic single-stranded DNA that can bind to and cleave complementary strands of a DNA substrate at the point of a designed ribonucleic base “impurity”. When a spider is placed on a surface on which the appropriate DNA substrate has been deposited (or nanoassembled), the legs bind to the substrate and catalyze its cleavage, creating two product strands. The upper portion floats away in solution and we do not consider it further. The lower portion remains on the surface, and, because it is complementary to the lower part of the leg, there is some residual binding of



the leg to the product, typically much weaker and shorter-lived than the leg-substrate binding. The leg kinetics are described by the five reactions in Eq. 1 relating legs (L), substrates (S), and products (P), in which we have folded the catalysis reaction and subsequent dissociation reactions into a single  $k_{\text{cat}}$  rate:



## 2.1 The Antal-Krapivsky Model

The Antal-Krapivsky model [2, 1] is a high-level abstraction. It represents molecular spiders as a (very uncommon kind of) random walker. Each walker has  $k$  legs (in the following results,  $k = 2$ ), whose chemical activity is independent, but whose motion is constrained by their attachment to a common body; in the model, any two legs must be within distance  $s$  (in the following results,  $s = 2$ ). The legs walk over sites on a regular 1D lattice, where each site is either a substrate or a product.

Mathematically, the AK model takes the form of a continuous-time Markov process, where the states of the system are given by the state of the lattice sites, and the state of the walker legs. All lattice sites are initially substrates and are only transformed to products when a leg detaches from the substrate (via catalysis). Thus the state of the lattice sites can be defined by the set  $P \subset \mathbb{Z}$  of product sites. The state of the walker is completely defined by the set  $F$  of attached feet locations. Thus any state can be described as the pair  $(P, F)$ .

We call  $F$  a *configuration* of the legs. The *gait* of a spider is defined by what configurations and what transitions between configurations are allowed in the model. In any state  $(P, F) \in \Omega$ , all  $k$  legs are attached. Together with the restriction that at most one leg may be attached to a site, this implies that

$$|F| = k. \tag{2}$$

Additionally, the legs are constrained by their attachment to a common body. If the spider has a point body with flexible, string-like legs of length  $s/2$ , then any two feet can be separated by at most distance  $s$ , thus

$$\max(F) - \min(F) \leq s. \tag{3}$$

The transitions in the process correspond to individual legs unbinding and re-binding. When a spider is in configuration  $F$ , any foot  $i \in F$  can unbind and move to a nearest-neighbor site  $j \in \{i + 1, i - 1\}$  to form a new configuration  $F' = (F \setminus \{i\}) \cup \{j\}$ , provided the new configuration does not violate one of the constraints of Eqs. 2 and 3. A transition  $i \rightarrow j$  is called *feasible* if it meets these constraints. The feasible transitions determine the gait of the spider. The

nearest-neighbor hopping combined with the mutual exclusion of legs leads to a shuffling gait, wherein legs can slide left or right if there is a free site, but legs can never move over each other, and a leg with both neighboring sites occupied cannot move at all. If the legs of such a spider were distinguishable, they would always remain in the same left-to-right ordering.

The rate at which feasible transitions take place depends on the state of the site  $i$ . If  $i$  is a product the transition rate is 1, but if  $i$  is a substrate the transition occurs at a slower rate  $r < 1$ . This is meant to model the realistically slower dissociation rates from substrates, corresponding to chemical kinetics where  $k_{\text{cat}}/k_{\text{p}}^- = r < 1$ . The effect of substrate cleavage is also captured in the transition rules. If for state  $(P, F)$ , where  $i \in F \setminus P$ , the process makes the feasible transition  $i \rightarrow j$ , then the leg will cleave site  $i$  before leaving, and the new state will have  $P' = P \cup \{i\}$ .

The relation of the AK model to the chemistry of the spiders in Eq. 1 can be understood if one assumes the chemical rates are given as in Eq. 4:

$$\begin{aligned} k_{\text{S}}^+ &= k_{\text{P}}^+ = \infty \\ k_{\text{S}}^- &= 0 \\ k_{\text{P}}^- &= 1 \\ k_{\text{cat}} &= r < 1 \end{aligned} \tag{4}$$

The infinitely fast on-rates account for all legs always being attached; when a leg unbinds it will instantly rebind to some neighboring site. Thus the spider is modeled as jumping from configuration  $F$  to configuration  $F'$ .

## 2.2 Superdiffusive Motion of Single AK Spiders

To characterize the motion of spiders we use the notion of *superdiffusion*. Superdiffusive motion can be quantified by analyzing the mean squared displacement (MSD) of a spider as a function of time. For diffusion in a one-dimensional space with diffusion constant  $D$ , the mean squared displacement is given by Eq. 5.

$$\text{msd}(t) = 2Dt^\alpha \begin{cases} \alpha = 0 & \text{stationary} \\ 0 < \alpha < 1 & \text{subdiffusive} \\ \alpha = 1 & \text{diffusive} \\ 1 < \alpha < 2 & \text{superdiffusive} \\ \alpha = 2 & \text{ballistic or linear} \end{cases} \tag{5}$$

We say that the spider is moving *instantaneously superdiffusively* at a given time  $t$  if

$$\alpha(t) = \frac{d(\log_{10} \text{msd}(t))}{d(\log_{10} t)} > 1. \tag{6}$$

Using Kinetic Monte Carlo simulations [4] of the Markov process we can estimate the MSD for the spider process for different parameter values by averaging over

many realizations  $x(t)$  of the process  $X(t)$ , where each  $x(t)$  is a function from  $t \in [0, t_{\max}]$  to the state space  $\Omega$  of the walker process, and  $x(t) \sim X(t)$ .

When  $r < 1$ , each spider process goes through three different regimes of motion defined by their instantaneous value for the exponent  $\alpha$  of  $\text{msd}(t)$ . Initially spiders are at the origin and must wait for both legs to cleave a substrate before they start moving at all and when  $t < 1/r$  the process is essentially stationary; we call this largely unimportant period the *initial period*. After the spiders take several steps, walkers with  $r < 1$  show a sustained period of superdiffusive motion over many decades in time. We call this the *superdiffusive period*, and define it as the period during which the instantaneous estimate of  $\alpha > 1.1$ . The cutoff of 1.1 is somewhat arbitrary but represents a threshold where spiders are moving significantly superdiffusively, in contrast to spiders with  $r = 1$ , which never have  $\alpha > 1$ . Finally, all spiders as predicted by Antal and Krapivsky will decay to an ordinary diffusion with  $\alpha \approx 1$ . This is called the *diffusive period* and is characterized by walkers mainly moving over regions of previously cleaved products, which makes the values of  $r$  irrelevant, since all walkers move with rate 1 over product sites.

To explain this behavior we observe that spiders with  $s = 2$  and  $k = 2$  always cleave all sites they move over since the AK model does not permit legs to change their effective ordering on the surface, and hence the walkers move with a shuffling gait consuming all products in the region they move over. This leads to the formation of a sharp boundary between a contiguous region of products called the *product sea*, and the remainder of the unvisited sites which are still substrates. The product sea has boundaries on either end, and only when the spider is at one of the boundaries can it be attached to substrates and hence affected by the parameter  $r$ .

To explain this behavior, we consider *boundary* and *diffusive* states. When the spider is in the boundary state, it moves ballistically towards unvisited sites; when it is in the diffusive state, its motion is ordinary diffusive. The transitions from the boundary state to diffusive state are independent of the previous state of the system before it entered the boundary state. However, the transitions from the diffusive state back to the boundary state depend on the size of the product sea that the spider has left behind, and this size increases with time. This explains the apparent superdiffusion at short times when the spider spends more time in the boundary state, and the decay to ordinary diffusion at long times, as the spider spends more and more of its time in the diffusive state.

There are two options to increase the superdiffusive effect of the spider motion: (1) decrease the effective size of the product sea, and hence the time needed to escape from it and return to the boundary; or (2) decrease the rate at which spiders leave the boundary. Here we focus on option (1), by means of localized release of spiders at the origin, which effectively fills the product sea with follower spiders, preventing the leading spiders from moving too far backwards away from the boundary. This works because spider legs cannot occupy the same site at the same time, and spiders walk with a shuffling gait, sliding left or right one site at a time, so a spider cannot jump over an adjacent spider. Thus, the

presence of multiple spiders will restrict the motion of the spiders around them, potentially reducing the effective size of the product sea as seen by a walker at the boundary. Thus by releasing many spiders sequentially at the origin we can make the diffusive state of the leading spiders shorter and perhaps make their transition into the boundary state independent of the number of sites they have already cleaved; such a system would have the potential for asymptotically superdiffusive motion.

### 3 Multiple Spiders Model

Here we consider a system with multiple  $k$ -legged spiders. The surface remains exactly the same as in the AK model. Thus the state of the system can now be described as  $X = (P, F_1, F_2, \dots, F_N)$  where, by analogy with the AK model,  $F_i \subset \mathbb{Z}$  is a set describing the state of the  $i$ th spider and  $N$  is the total number of spiders released onto the surface. As new spiders are released  $N$  grows with time. All spiders are exactly the same and so parameters  $k = 2$  and  $s = 2$  apply to all spiders:

$$|F_i| = k, \text{ for all } i. \quad (7)$$

$$\max(F_i) - \min(F_i) \leq s, \text{ for all } i. \quad (8)$$

To extend the chemical exclusionary properties of spider legs to multi-spider systems, we add the restriction that any site on the surface can be occupied by only one leg of any spider:

$$F_i \cap F_j = \emptyset, \text{ for all } i, j. \quad (9)$$

With multiple spiders on a single lattice, there are situations where a particular spider is completely blocked from movement when other spiders occupy the sites to its immediate left and right. Thus, to simplify the Markov process description, we introduce a slight change to the gait of the walker with respect to the AK model. When a leg detaches from a site  $i$  it can move not only to sites  $i - 1$  and  $i + 1$ , but also back to site  $i$ . It chooses from any site in  $\{i - 1, i, i + 1\}$  with equal probability, provided none of the new configurations violates the constraints of Eqs. 7, 8, and 9. Thus, even if sites  $i - 1$  and  $i + 1$  are occupied, the leg has somewhere to go. This change of the gait also makes the model more realistic, as the enzymatic leg of a real spider can always rebind to the site it just dissociated from.

All legs of all spiders move independently. As with a single spider, the constraints enforce that a leg is never detached for a finite amount of time, i.e., legs hop to neighboring sites or step back onto the previous site infinitely fast.

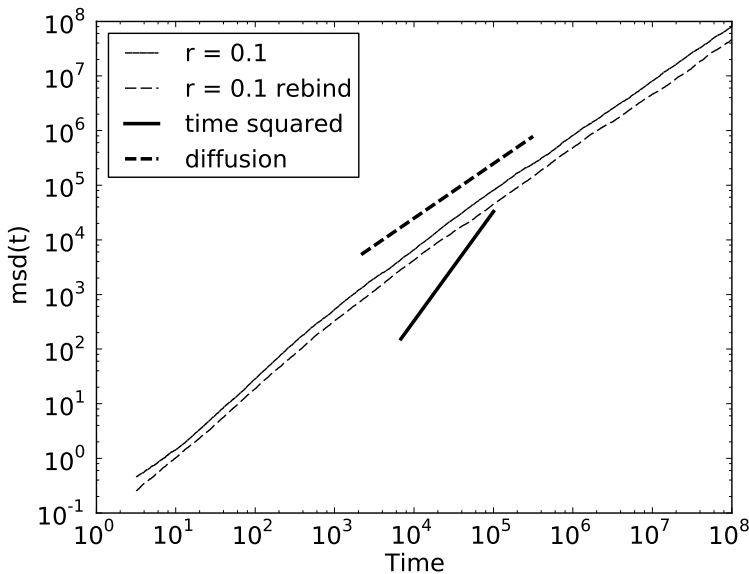
We start all experiments from a symmetric configuration with two spiders placed on the surface, one left of the origin with legs at  $\{-3, -1\}$  and the other right of the origin with legs at  $\{1, 3\}$ . The initial set of products is  $P = \{-2, -1, 0, 1, 2\}$ ; all other sites are substrates. The pair of sites  $\{0, 1\}$  is the injection point for new spiders. A new spider is released whenever the injection point is unoccupied.

## 4 Simulation Results for Multiple Spiders

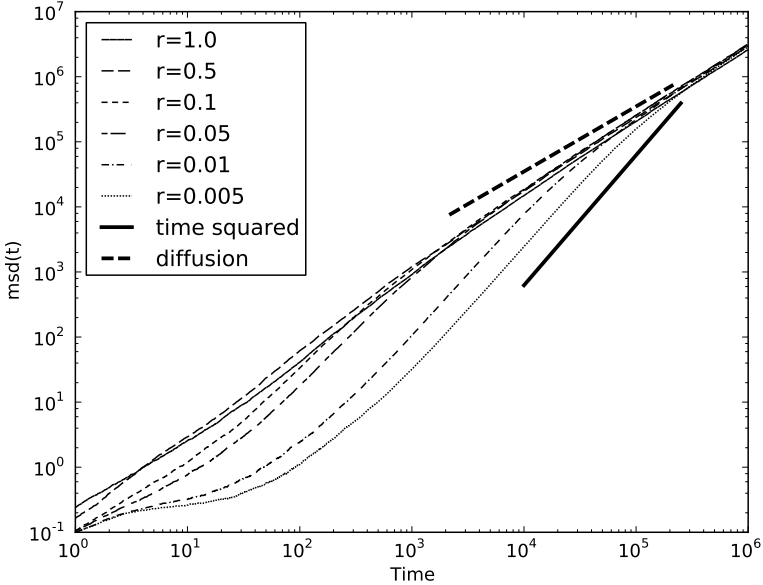
Similarly to our single-spider experiments [13] we use the Kinetic Monte Carlo method [4] to numerically sample traces of the multi-spider Markov process. We vary the chemical-kinetics rate  $r$  to see how it influences the motion. In case when  $r = 1$  there is no effective difference between substrates and products.

### 4.1 Comparison of a Single AK Model Spider with a Single Spider of the Current Model

Recall from Sec. 3 that the action of individual spiders in the multi-spider model was modified to incorporate the possibility of a leg rebinding to the site it just detached from. This is a realistic modification of the model for single spiders, but as it represents a formal model change, we investigate its effects on the motion of single walkers. We compare the results of the AK model with the modified model permitting rebinding through KMC simulations, using  $k = 2$ ,  $s = 2$ , and  $r = 0.1$ , and show the  $\text{msd}(t)$  estimates in Fig. 1. These results show substantially the same qualitative behavior. However, spiders with rebinding move at a constant-factor slower pace than the original model, as the transitions which lead to a rebinding do not move the walker in any direction. This effectively slower rate needs to be taken into account when comparing the motion of spiders in the multi-spider model to those in the AK model, but it does not fundamentally change anything about the characteristics of spider motion.



**Fig. 1.** Comparison of  $\text{msd}(t)$  for a single spider moving in the AK model and with the modified model permitting rebinding of a leg to its previous site

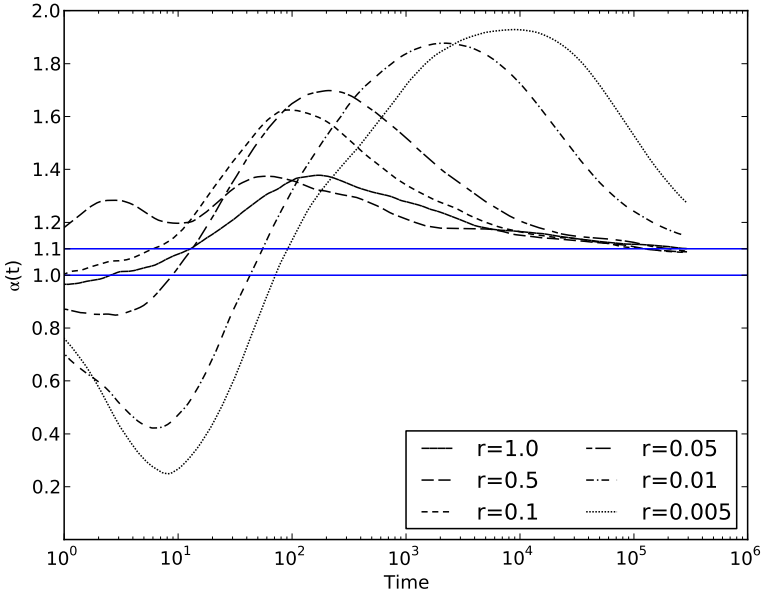


**Fig. 2.** Mean squared displacement for the leading spider in multi-spider simulations. Reference lines are shown for ordinary diffusion and ballistic motion.

## 4.2 Multiple Spiders Simulations

For each value of  $r \in \{1, 0.5, 0.1, 0.05, 0.01, 0.005\}$  we used our KMC algorithm to simulate 1200 independent realizations of the multi-spider model described in Sec. 3. We used these realizations to sample the number of spiders and the position of each of the spiders at regularly spaced time intervals. We define the position of a spider to be the mean of its attached leg positions (i.e.,  $\sum F_k/2$  for spider  $k$ ). To ensure that each simulation trace provides a sample for each measured time, we run each simulation until time is at least  $t_{\max} = 10^6$ . We choose the measurement time points to be equispaced for the independent axis of the plots reported, so that for linear plots the successive time intervals have a constant difference, and for log axes, the successive time intervals have a constant ratio.

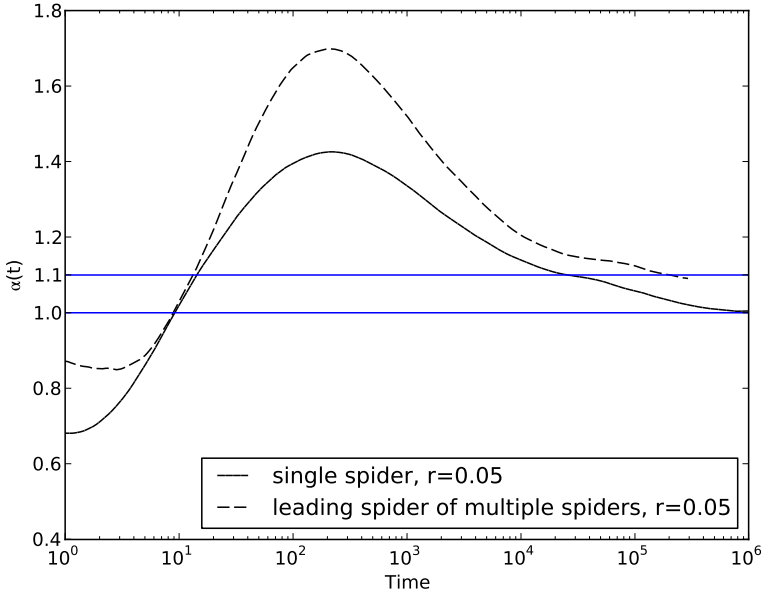
**Observed Superdiffusion of the Leading Spiders.** As discussed in Sec. 2.2, it is known that single spiders show transient superdiffusive behavior [13]. Single spiders with  $r < 1$  move faster than ordinary diffusion for a significant time and distance, but eventually slow down and move as an ordinary diffusion. The leading spiders in the multi-spider model also initially move superdiffusively, but they reach higher values of  $\alpha$  and thus they are closer to ballistic motion than single spiders at peak times; furthermore, the decay towards ordinary diffusion appears to be incomplete. Fig. 2 shows the estimate of  $\text{msd}(t)$  for the leading spider on a log-log plot for each measured  $r$  parameter value. In this plot, straight



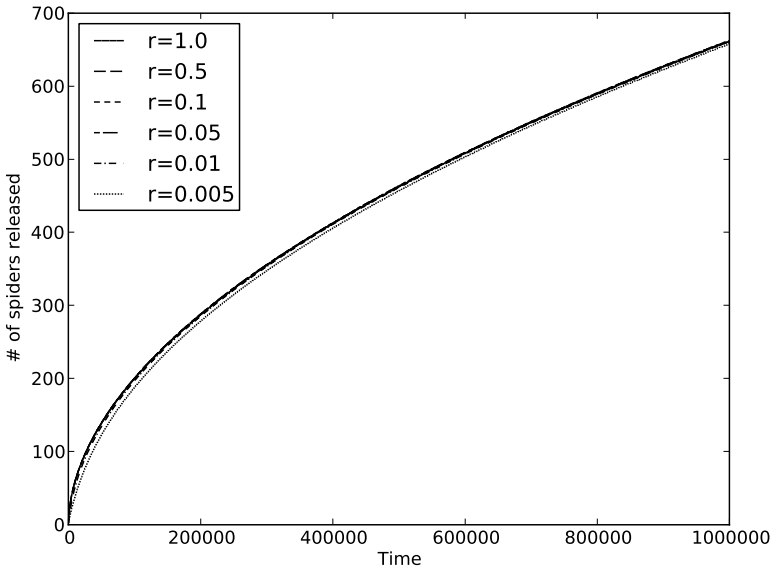
**Fig. 3.** Finite difference approximation of  $\alpha(t)$  for the leading spider in multi-spider simulations. Horizontal lines define the threshold for ordinary diffusion at  $\alpha = 1$  and our defined threshold for superdiffusion at  $\alpha = 1.1$ .

lines correspond to power laws, that is, to Eq. 5, and the parameter  $\alpha$  is given by the slope. To show the instantaneous value of  $\alpha$ , we use finite difference methods to estimate  $\alpha(t)$  (Eq. 6), and Fig. 3 shows the result of using the Savitzky-Golay smoothing filter [11] on these estimates. Fig. 4 gives a comparison of  $\alpha(t)$  for the single spider of the AK Model and the leading spider of the multi-spiders model.

In Sec. 2.2 we explained that single spiders in the AK model have been observed to have three distinct regimes of motion defined by their value of  $\alpha(t)$ . There is an initial stationary regime before the walker starts moving, followed by a superdiffusive regime spanning many decades in time, and finally a diffusive regime as  $\alpha(t)$  falls back to 1, where the walkers spend most of their time diffusing in the product sea. In contrast, for the leading spiders in the multi-spider model, the third regime is a gradual but incomplete shift towards diffusion. There is a decrease in  $\alpha(t)$  at longer times (Fig. 3), but, at least within the simulated time bound of  $t_{\max} = 10^6$ , the motion remains superdiffusive. Unlike for the single-spider model, there are as yet no analytical results for the multi-spider model. The true behavior at greater times thus remains a matter for speculation, but we do notice a clearly different behavior than for single spiders within the simulated times (Fig. 4). Also note that, as in the single-spider model, decreasing values of  $r$  lead to increasingly superdiffusive behavior, but, unlike in the single-spider model, even spiders with  $r = 1$  move superdiffusively. This is not unexpected, because even when  $r = 1$  there is an exclusionary pressure exerted on the outermost spiders that prevents them from returning to the origin as would occur for

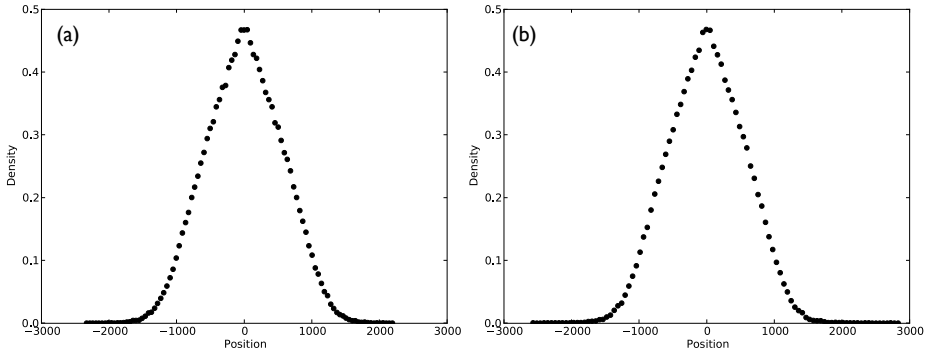


**Fig. 4.** Comparison of finite difference approximation of  $\alpha(t)$  for the leading spider in multi-spider simulations and the single spider in simulations of the AK model

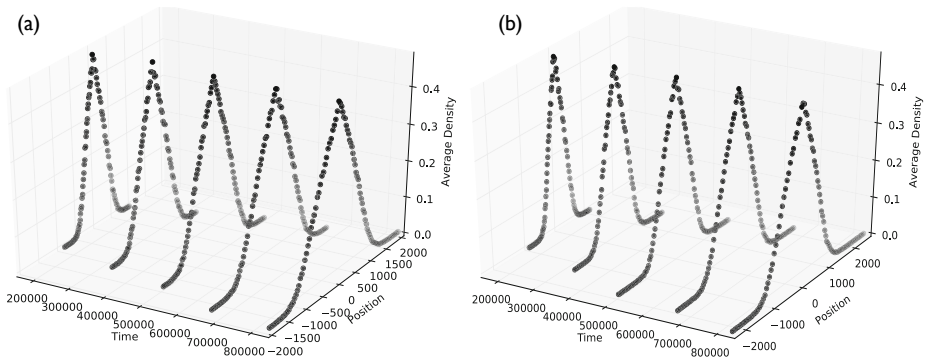


**Fig. 5.** Number of released spiders for multi-spider simulations





**Fig. 6.** Average spider density at  $t_{\max}$ , for  $r = 1$  (a) and for  $r = 0.05$  (b)



**Fig. 7.** Average spider density plotted at several instants, for  $r = 1$  (a) and for  $r = 0.05$  (b)

single walkers with  $r = 1$ . However, as with the single spiders, when  $r < 1$  we see a much enhanced superdiffusive effect, something not possible for a walker that does not transform sites irreversibly like the molecular spiders.

**Number of Released Spiders.** Spiders in the multi-spider model are released at the origin *whenever possible*, so the number of spiders actually released by time  $t$  is a random variable of interest, and estimates for its mean are shown in Fig. 5 for each studied value of  $r$ .

We observe that the average number of spiders grows sublinearly. Thus, attempts to release spiders are often unsuccessful, because of interference from other spiders at the origin. This indicates that in one dimension spiders move away from the origin relatively slowly.

**Density of Spiders.** The density of spiders gives some insight into why the leading spiders move superdiffusively, even for  $r = 1$ , and why the number of spiders added does not grow linearly. We measure the density of spiders as the

average probability for each site to be occupied by a spider at a particular time. Shown in Fig. 6 is the spider density at time  $t_{\max}$ . Clearly, spiders with  $r = 0.05$  have spread out slightly farther, but both  $r$ -values show a much higher density of spiders around the origin where new spiders are released. The evolution of this density through time can be seen in Fig. 7.

## 5 Discussion

Our analysis of the multi-spider model shows significant differences from the previous work on single spiders. The most fundamental difference is that (at least within the times simulated) walkers for all values of  $r$  move superdiffusively with  $\alpha(t_{\max}) > 1.1$ . However, as with the single-spider model, decreasing values of  $r$  lead to increasingly superdiffusive behavior. This is an essential property of molecular spiders that distinguishes them from many other types of molecular walkers. The spider superdiffusion depends on there being a residency-time bias between visited and unvisited sites (i.e.,  $r < 1$ ) and it also depends on the walkers having more than one leg. Thus, even a single spider shows some cooperative behavior between the two legs to enable an emergent superdiffusive effect. However, the interactions in the multi-spider model show an even more significant effect, and this can be attributed to the cooperative collective behavior of the swarm of walkers. The furthest walkers from the origin do all of the cleaving of sites, but the internal walkers act to exert a “pressure” on the outermost walkers, preventing them from moving backwards too far, and keeping their behavior superdiffusive.

There are many possibilities for adding stronger interactions between spiders that will potentially lead to even more pronounced emergent behaviors. However, the present work shows that even simple interactions, defined solely by an exclusion property that prevents multiple walkers from binding to the same site at once, can lead to motion that is faster than diffusion, at least over the finite times simulated. These results can be used to design collective spider transport systems that can perform useful tasks at the nanoscale.

**Acknowledgments.** This material is based upon work supported by the National Science Foundation under grant 0829896.

## References

1. Antal, T., Krapivsky, P.L.: Molecular spiders with memory. *Physical Review E* 76(2), 021121 (2007)
2. Antal, T., Krapivsky, P.L., Mallick, K.: Molecular spiders in one dimension. *Journal of Statistical Mechanics: Theory and Experiment* 2007(08), P08027 (2007)
3. Badoual, M., Julicher, F., Prost, J.: Bidirectional cooperative motion of molecular motors. *PNAS* (10), 6696–6701 (2002)
4. Bortz, A.B., Kalos, M.H., Lebowitz, J.L.: A new algorithm for Monte Carlo simulation of Ising spin systems. *Journal of Computational Physics* 17(1), 10–18 (1975)

5. Campas, O., Kafri, Y., Zeldovich, K.B., Casademunt, J., Joanny, J.F.: Collective dynamics of interacting molecular motors. *Physical Review Letters* 97, 038101 (2006)
6. Frey, E., Parmeggiani, A., Franosch, T.: Collective phenomena in intracellular processes. *Genome Informatics* 15(1), 46–55 (2004)
7. Hirokawa, N., Takemura, R.: Molecular motors and mechanisms of directional transport in neurons. *Nature Reviews: Neuroscience* (6), 201–214 (2005)
8. Julicher, F., Ajdari, A., Prost, J.: Modeling molecular motors. *Reviews of Modern Physics* 69(4), 1269–1281 (1997)
9. Lund, K., Manzo, A.J., Dabby, N., Michelotti, N., Johnson-Buck, A., Nangreave, J., Taylor, S., Pei, R., Stojanovic, M.N., Walter, N.G., Winfree, E., Yan, H.: Molecular robots guided by prescriptive landscapes. *Nature* 465, 206–210 (2010)
10. Pei, R., Taylor, S.K., Stefanovic, D., Rudchenko, S., Mitchell, T.E., Stojanovic, M.N.: Behavior of polycatalytic assemblies in a substrate-displaying matrix. *Journal of the American Chemical Society* (128), 12693–12699 (2006)
11. Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: *Numerical recipes in C++*. Cambridge University Press, New York (2002)
12. Schliwa, M., Woehlke, G.: Molecular motors. *Nature* (422), 759–765 (2003)
13. Semenov, O., Olah, M.J., Stefanovic, D.: Mechanism of diffusive transport in molecular spider models. *Phys. Rev. E* 83(2), 021117 (2011)

# Author Index

- Arnold, Mark G. 34
- Brijder, Robert 49
- Chandran, Harish 64  
Condon, Anne 84  
Czeizler, Eugen 145
- Danos, Vincent 1  
Demaine, Erik D. 100
- Eisenstat, Sarah 100
- Gillis, Joris J.M. 49  
Gopalkrishnan, Nikhil 64
- Hu, Alan 84
- Ishaque, Mashhood 100
- Koepl, Heinz 1  
Konstantinidis, Stavros 115  
Krishnan, Yamuna 22
- Lakin, Matthew R. 130  
Lehman, Niles 32  
Lempiäinen, Tuomo 145  
Lutz, Jack H. 21
- Mañuch, Ján 84  
Mellor, Jessica 32  
Modi, Souvik 22
- Olah, Mark J. 160, 204  
Orponen, Pekka 145
- Patitz, Matthew J. 175  
Phillips, Andrew 64, 130
- Reif, John 64  
Rodríguez-Patón, Alfonso 190
- Sainz de Murieta, Iñaki 190  
Santean, Nicolae 115  
Schweller, Robert T. 175  
Semenov, Oleg 204  
Sosík, Petr 190  
Stefanovic, Darko 160, 204  
Summers, Scott M. 175  
Surana, Sunaina 22
- Thachuk, Chris 84
- Vaidya, Niles 32  
Van den Bussche, Jan 49
- Wilson-Kanamori, John 1  
Winslow, Andrew 100
- Yan, Hao 33